Proyecto Fin de Carrera
Ingeniería de Telecomunicación

Plataforma para la gestión de pacientes con medicamentos anticoagulantes

Autora: Keyla Cruz Vera
Tutor: José Manuel Fornés Rumbao

Dpto. de Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2021

Proyecto Fin de Carrera
Ingeniería de Telecomunicación

Plataforma para la gestión de pacientes con medicamentos anticoagulantes

Autora:
Keyla Cruz Vera

Tutor:
José Manuel Fornés Rumbao
Profesor titular

Dpto. de Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2021

Trabajo Fin de Grado: Plataforma para la gestión de pacientes con medicamentos anticoagulantes

Autora: Keyla Cruz Vera

Tutor: José Manuel Fornés Rumbao

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2021

El Secretario del Tribunal

A mi familia

Agradecimientos

A mi madre, por ser esa persona que me inspira cada día, aquella que luchó con todas sus fuerzas en los momentos más difíciles de nuestras vidas y que se plantó con valentía y coraje para sacarnos adelante y no dejarnos caer. Me siento afortunada por tenerla, le agradezco haberme apoyado siempre en todas mis decisiones y también por basar mi crecimiento en la consciencia de que las cosas cuestan y hay que trabajar duro en la vida para conseguirlas, que rendirse no cuenta como opción.

A mi esposo, un ser de luz que por suerte un día se cruzó en mi camino y que desde entonces me acompaña y me apoya incondicional y desinteresadamente. Aquel con el que comparto absolutamente todo, mis pensamientos, mis sentimientos, mis risas y mis llantos, mi vida en general, aquel con el que soy un libro abierto y aquel que en momentos en los que yo casi perdía la confianza de si podría lograr llegar hasta aquí, ha creído en mí y otra vez me ha empujado a intentarlo.

Sin duda ha sido todo un reto en el que he podido experimentar un montón de sensaciones, esos cuadros de ansiedad momentos antes de un examen, frustración si no conseguía entender algo, desmotivación por aquellos largos periodos de exámenes encerrada de sol a sol en una biblioteca o incertidumbre por esas largas esperas hasta que llegaba una nota, pero no puedo negar esa inmensa alegría y satisfacción que he ido sintiendo aprobado tras aprobado viendo como poco a poco y con esfuerzo iba acercándose este momento. Sin duda, todo ha merecido la pena y estoy orgullosa de haberlo conseguido.

Keyla Cruz Vera

Sevilla, 2021

Resumen

El desarrollo web es una de las materias que ha crecido de manera exponencial sobre todo en las últimas décadas. Cualquiera de nosotros es consciente de la cantidad de horas al día que dedicamos a navegar por internet, por tanto, podemos decir que el desarrollo web ha transformado la forma en la que vivimos facilitándonos el intercambio de información a nivel global. Hoy en día casi cualquier búsqueda de información, trámite o gestión tiene como primera opción abrir nuestro navegador.

A medida que pasa el tiempo se escalan nuevos pasos trabajando para conseguir un ambiente web más estable, creativo y eficiente donde la interacción del usuario con la red, sea sencilla y práctica, en donde la prioridad es la información y el uso de esta, creando métodos de intercambio rápidos y eficaces, para llevar a cabo proyectos no solo de lectura o consulta de información, sino también herramientas para la gestión de la información.

Anteriormente en las páginas y aplicaciones todo estaba construido en un solo paquete. La página web se construía completa en el backend o servidor y se enviaba al frontend o navegador para presentarse. Con esto cada vez que se requería cierta información, el navegador realizaba una petición al servidor y se reconstruía la página completa lo cual recargaba todo el navegador y esto era muy lento.

En la actualidad el backend y frontend son desarrollos completamente separados y ambos fuertemente robustos, que se comunican entre sí de manera asíncrona por medio de APIs REST.

Los Frameworks de javascript modernos impulsan la creación de aplicaciones de una sola página SPA (Single Page Application) que no recargan el navegador, sino que solo refrescan la información que se requiere, por lo que son mucho más rápidas y se le brinda al usuario una experiencia más fluida.

Todo esto también nos permite tener sistemas más escalables ya que toda la lógica y programación del backend puede ser reutilizada para otras interfaces o aplicaciones.

Este proyecto aborda la implementación de un servidor basado en API REST que presta servicio a:

- a una aplicación web creada para que un médico pueda gestionar y llevar el seguimiento de pacientes bajo tratamientos anticoagulantes.
- a una aplicación móvil destinada a los pacientes para que puedan notificar eventos, consultar su historial y seguir instrucciones que su médico enviará a través de la aplicación web.

La implementación de la aplicación web para los médicos

Abstract

Web development is one of the technologies that has grown exponentially especially in the last decades. Any of us is aware of how many hours a day we spend surfing the Internet, therefore, we can say that web development has transformed the way we live facilitating the exchange of information globally. Nowadays almost any information search, procedure or management has as first option to open our browser.

As time goes by, new steps are being taken to achieve a more stable, creative and efficient web environment where the user's interaction with the network is simple and practical, where the priority is the information and its use, creating fast and efficient methods of exchange, to carry out projects not only for reading or consulting information, but also tools for information management.

In the past, websites and applications were all built in a single package. The web page was built completely in the backend or server and sent to the frontend or browser to be presented. With this, every time certain information was required, the browser made a request to the server and the entire page was rebuilt, which reloaded the entire browser and this was very slow.

Nowadays the backend and frontend are completely separate developments and both strongly robust, which communicate with each other asynchronously via REST APIs.

Modern javascript frameworks drive the creation of SPA (Single Page Application) single page applications that do not reload the browser, but only refresh the information that is required, so they are much faster and provide the user with a smoother experience.

All this also allows us to have systems that are more scalable since all the backend logic and programming can be reused for other interfaces or applications.

This project addresses:

- implementation of a backend based on REST API that serves:
    - web application created for a doctor to manage and keep track of patients on anticoagulant treatment
    - mobile application intended for patients so that they can report events, consult their history and follow instructions that their doctor will send through the web application.
- implementation of the web application for doctors.

Índice

ÍNDICE DE FIGURAS

1 ESTADO DEL ARTE

1.1 VISIÓN GENERAL

La idea principal de este proyecto es la implementación por separado de un servidor Rest y de un cliente web que interactuarán entre sí, es decir, dos implementaciones completamente independientes, de manera que no se mezcla el código de estos dos paradigmas como pasaba en las aplicaciones monolíticas clásicas.

Liberando esa dependencia entre ambos prototipos, se va a conseguir mejor mantenibilidad, escalabilidad y reusabilidad a largo plazo.

Este trabajo de fin de grado se ha estructurado de la siguiente manera:

- Diseño, desarrollo, implementación y securización de un servidor REST, capaz de servir a dos tipos de usuarios: médicos y pacientes. Además, se ha realizado la integración con Firebase para poder enviar notificaciones a una aplicación móvil (pacientes).
- Diseño, desarrollo e implementación de una aplicación web (médicos).

Este proyecto ha sido desarrollado con tecnologías modernas y punteras que hoy en día están en auge tanto para el lado del servidor como para el lado del cliente web, que tienen una gran comunidad activa de desarrolladores y usuarios y que tiene un largo y prometedor futuro.

A continuación, se pasará a describir cada una de las tecnologías implementadas

1.2 TECNOLOGÍAS IMPLEMENTADAS

1.2.1 Lado servidor de sitios web

El servidor Rest o Api Rest ha sido desarrollado con Node.js y securizado mediante JWT (JSON Web Token) que es un estándar abierto basado en JSON (JavaScript Object Notation), propuesto por IETF (RFC 7519) para la creación de tokens de acceso.

1.2.1.1 Node.js

Node.js es un entorno de ejecución para JavaScript, de código abierto, multi-plataforma, construido con el motor de JavaScript V8 de Chrome, orientado a eventos asíncronos.

Figura 1-1. Node.js - entorno de ejecución para JavaScript

JavaScript era un lenguaje exclusivamente del lado del cliente, que solo podía ser ejecutado sobre un navegador web.

Sin embargo, esto dejó de ser así en 2009 cuando Ryan Dhal, presentó un entorno de ejecución con todo lo necesario para ejecutar JavaScript fuera de un navegador web.

Esto fue posible gracias al uso de un revolucionario un motor de JavaScript, de código abierto, desarrollado por Google para su navegador Chrome. Se trata de V8.

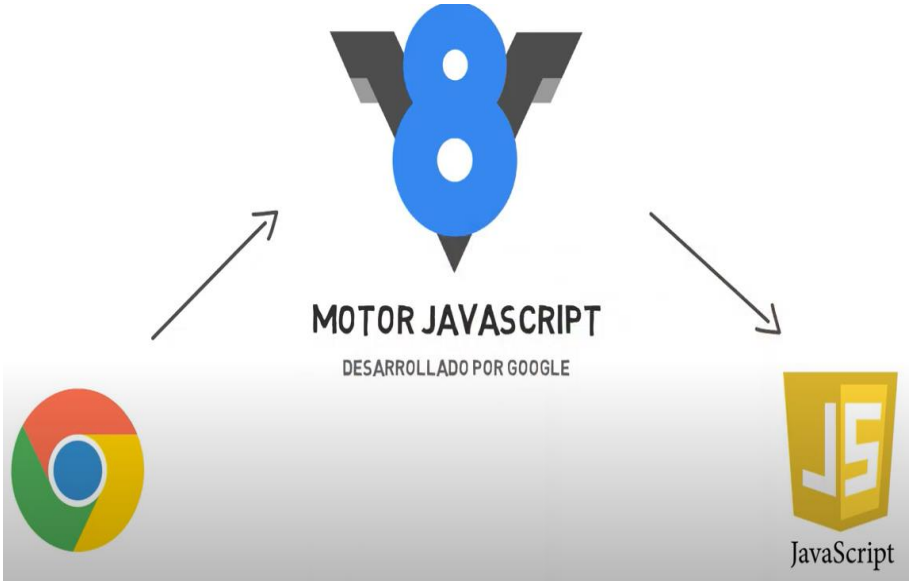


Figura 1-2.. Motor de JavaScript V8 de Chrome

Este motor, puede ejecutar JavaScript a alta velocidad y de forma totalmente independiente a los navegadores.

Una característica importante, es que este entorno de ejecución llamado Node.js, usa un modelo de operaciones asíncronas y orientado a eventos.

Figura 1-3.. Modelo asíncrono de Node.js

Por lo tanto, a diferencia de un modelo operaciones síncronas, cuyas tareas se van ejecutando en función del orden en que hayan sido programadas, en el modelo asíncrono, el procesamiento de una tarea no va a bloquear o retrasar la ejecución de otra, una característica que lo hace muy liviano y eficiente a la hora de soportar aplicaciones altamente escalables, con grandes cantidades de datos de entradas/salidas o las aplicaciones de tiempo real.

Node.js también cuenta con su propio gestor de paquetes, llamado NPM (Node Package Manager).

Figura 1-4. NMP - Gestor de paquetes para Node.js

Mediante este ecosistema de librerías, podemos gestionar todas las dependencias necesarias para nuestro proyecto

Ventajas

- Ha sido diseñado para optimizar el rendimiento y la escalabilidad en aplicaciones web.
- Puede atender muchas conexiones simultáneamente, por lo que goza de gran rendimiento.
- El código está escrito en JavaScript, que es un lenguaje de programación relativamente nuevo y se beneficia de los avances en diseño de lenguajes cuando se compara con otros lenguajes de servidor web tradicionales como Python, PHP, etc.
- El gestor de paquetes de Node.js es NPM (Node Packet Manager) y proporciona acceso a miles de paquetes reutilizables.
- Es portable, con versiones que funcionan en multitud de sistemas operativos entre ellos: Windows, Mac OS X, Linux, Solaris, FreeBSD, OpenBSD, WebOS, entre otros.
- Está altamente soportado por muchos de los proveedores de hospedaje web, que proporcionan infraestructura específica y documentación para hospedaje de sitios Node.js.
- Tiene un ecosistema y comunidad de desarrolladores muy activa.

1.2.1.2 API REST - Protocolo de comunicación

Antes de introducir el concepto de API Rest, se explica brevemente que una API (Interfaz de programación de aplicaciones) es un conjunto de definiciones y protocolos que se utiliza para permitir la comunicación de datos entre dos aplicaciones a través de un conjunto de reglas.

Para ello, la API utiliza solicitudes HTTP (Hypertext Transfer Protocol), protocolo orientado a transacciones que sigue el esquema petición-respuesta entre un cliente y un servidor.

Las principales solicitudes son:

- **GET:** Es utilizado únicamente para **consultar información** al servidor, muy parecidos a realizar un SELECT a la base de datos. No soporta el envío del body en la petición, sin embargo, si necesitamos una consulta con filtro, los parámetros a filtrar se pueden enviar a través de la url.
- **POST:** Es utilizado para solicitar la **creación de un nuevo registro**, es decir, algo que no existía previamente. Es comparable a realizar un INSERT en la base de datos. Los datos del registro a crear se envían por el body
- **PUT**: Se utiliza para **actualizar por completo un registro existente**, es decir, es parecido a realizar un UPDATE a la base de datos. Los datos del registro a actualizar se envían por el body
- **PATCH**: Este método es similar al método PUT ya que también sirve para actualizar un registro existente, sin embargo, este se utiliza cuando se va a **actualizar el registro parcialmente** y no en su totalidad, también sería equivalente a realizar un UPDATE a la base de datos y los datos del registro a actualizar se envían por el body
- **DELETE**: Este método se utiliza para **eliminar un registro existente**, es similar a DELETE a la base de datos. No soporta el envío de parámetros a través del body de la petición.

Los códigos de respuestas HTTP más habituales que podemos recibir a nuestra petición son:

2xx – Indican que la petición se ha satisfecho correctamente

- 200 - La solicitud ha tenido éxito y se ha encontrado el recurso y procesado la respuesta.
- 201 - La solicitud ha tenido éxito y se ha creado un nuevo recurso como resultado de ello.
- 204 - La petición se ha completado con éxito, pero su respuesta no tiene ningún contenido.

4xx – indican códigos de error en el lado del cliente

- 400 - Esta respuesta significa que el servidor no pudo interpretar la solicitud dada una sintaxis inválida.
- 401 - Para decirnos que no tenemos autorización para ese recurso ya que es necesario autenticar para obtener la respuesta solicitada
- 404 - El servidor no pudo encontrar el contenido solicitado. Este código de respuesta es uno de los más famosos dada su alta ocurrencia en la web.
- 405 - Que nos indica que el método no se admite en la API.

5xx- indican códigos de error en el lado del servidor

- 500 - Cuando ha habido algún error interno del servidor y es una situación que no sabe cómo manejarla.
- 502 - Cuando el servidor hace de Gateway o proxy y ha recibido una respuesta de error a donde quiera que intentase conectarse.
- 503 - Cuando el servicio en el servidor no está accesible, normalmente debido a una sobrecarga o algún problema similar.

Un servicio **REST** (Representational State Transfer) es un conjunto de restricciones que las solicitudes HTTP tienen que cumplir según las directrices definidas en la arquitectura software.

Las restricciones que definen a un sistema REST son:

- **Cliente-servidor**: esta restricción se asegura de que las tareas estén perfectamente definidas entre el servidor y el cliente, el cliente manejará las correspondientes al interfaz de usuario y el servidor la del almacenamiento y gestión de datos. Gracias a esta arquitectura podemos tener un sistema que es capaz de gestionar clientes diferentes con interfaces distintos sin preocuparse de la presentación al usuario que se gestionará en el cliente (web, apps, móvil…)

- **Sin estado**: aquí decimos que cada petición que recibe el servidor debería ser independiente, es decir, el servidor no almacena ningún tipo de información o contexto (lo que se denomina **estado**) del cliente entre las distintas peticiones. El cliente será el responsable de gestionar el estado de sus sesiones, es decir la información que necesita tener accesible para la sesión. Cualquier petición realizada desde el cliente tendrá que contener toda la información que se necesite para poderla procesar en el servidor. Si el estado de la sesión es relevante para la petición, como por ejemplo las variables para la autenticación de la petición, se tendrá que incluir en la petición al servidor.

- **Cacheable**: Cada respuesta a una petición deberá indicar si se puede cachear en el cliente o no y por qué periodo de tiempo. La capacidad de cachear información en el lado del cliente es una parte esencial del rendimiento de la arquitectura ya que evitará repetir varias conexiones entre el servidor y el cliente para recuperar un mismo recurso.

- **Interfaz uniforme**: define una interfaz genérica para administrar cada interacción que se produzca entre el cliente y el servidor de manera uniforme, lo cual simplifica y separa la arquitectura. Esta restricción indica que cada recurso del servicio REST debe tener una única dirección, “URI”.

- **Sistema de capas**: para el cliente debería ser indiferente si está conectado directamente al servidor o se ha conectado a través de algún tipo de intermediario como un proxy o una Red de Distribución de Contenidos. Esto es un componente esencial para garantizar la escalabilidad, el rendimiento y la seguridad.

- **Interfaz uniforme**:
    - cada recurso del servicio REST debe tener una única dirección o URI (identificador del recurso uniforme).
    - Una vez que el cliente tiene la representación de un recurso podrá modificarlo o eliminarlo, siempre y cuando cuente con los permisos adecuados para acceder y editar el recurso en el servidor.
    - Tanto al enviar una petición como al recibir la representación de un recurso tenemos que indicar cual es el formato que usamos en cada caso. Sin esta información no se puede analizar los datos de manera confiable. Por ejemplo: si el formato de los datos es JSON, se indicaría

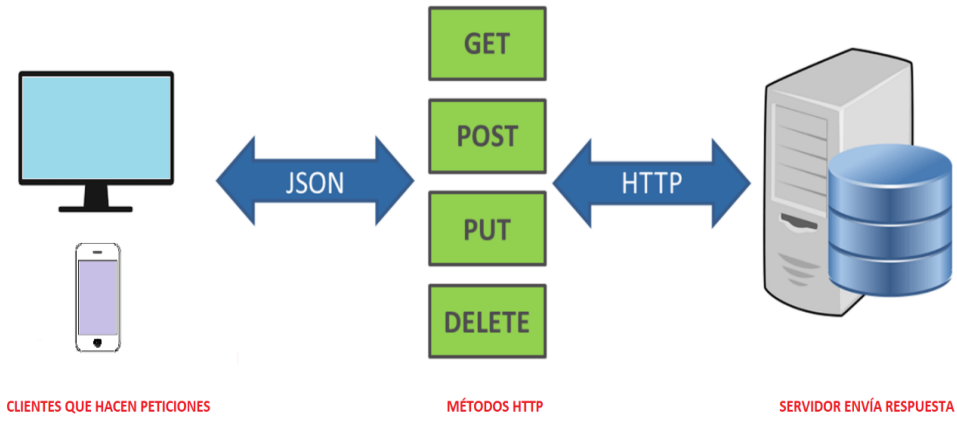


Figura 1-5. Comunicación entre clientes-servidor mediante protocolos HTTP

Por tanto, una **API REST**, va a realizar la comunicación de datos entre aplicaciones, respetando el estándar REST para las solicitudes HTTP.

Hoy en día la mayoría de las empresas utilizan API REST para crear servicios web. Esto se debe a que es un estándar lógico y eficiente.

Algunos ejemplos son los sistemas de identificación de Facebook o la autenticación en los servicios de Google.

Las API REST han sido diseñadas para simplificar el clásico WebService con XML (eXtensible Markup Language), ya que gracias al envío de datos por medio de JSON (JavaScript Object Notation) o su recepción por este medio podemos hacer solicitudes sencillas, rápidas e intuitivas.

1.2.1.3 JSON (JavaScript Object Notation)

JSON (JavaScript Object Notation) es un formato basado en texto estándar para representar datos estructurados en la sintaxis de objetos de JavaScript. Es comúnmente utilizado para transmitir datos en aplicaciones web.

Un JSON es una cadena cuyo formato recuerda al de los objetos literales JavaScript.

Es posible incluir los mismos tipos de datos básicos dentro de un JSON que en un objeto estándar de JavaScript - cadenas, números, arreglos, booleanos, y otros literales de objeto.

Esto permite construir una jerarquía de datos, como ésta:

```
{
      "nombre": "Keyla",
      "apellido": "Cruz",
      "ciudad": "Sevilla",
      "mascotas": [
        {
                "nombre": "Simba",
                "edad": 9,
                "genero": "masculino",
                "especie": "felino"
        },
        {
                "nombre": "Lola",
                "edad": 3,
                "genero": "femenino",
                "especie": "felino"
        }
      ]
}
```

1.2.1.4 Securización del servidor REST

La acción de securizar un servidor es un paso indispensable que sirve principalmente para proteger los datos que se manejan y dar permiso de interacción solamente a usuarios legítimamente autenticados.

Como primer paso vamos a analizar la principal diferencia entre un sistema de autenticación basado en JWT (JSON Web Token) frente al esquema tradicional basado en el uso de sesiones.

Los JWT proporcionan un medio para mantener el estado de la sesión en el cliente en lugar de hacerlo en el servidor.

Con las sesiones del lado del servidor, se tiene que almacenar el identificador de sesión en una base de datos, o bien mantenerlo en la memoria y asegurarse de que el cliente siempre llegue al mismo servidor.

Ambas opciones tienen inconvenientes.

- En el caso almacenar el id en una base de datos (u otro almacenamiento centralizado), esto se convierte en un cuello de botella y algo que mantener ya que como mínimo siempre se va a hacer una consulta adicional que se debe realizar con cada solicitud.
- Con una solución en memoria, se limita su escalabilidad horizontal, es decir, a largo plazo se complicaría la posibilidad de *tener varios servidores como un todo* y las sesiones se ven afectadas por problemas de red (clientes en itinerancia entre Wifi y datos móviles, reinicio de servidores, etc.).

Mover la sesión al cliente significa que elimina la dependencia en una sesión del lado del servidor

1.2.1.5 JWT (JSON Web Token)

JWT es un **protocolo de autenticación** basado en un estándar abierto (RFC 7519) en el cual se define un conjunto de operaciones o mecanismo para transmitir información con la identidad y permisos de un usuario de forma segura entre un cliente/servidor.

En la autenticación por token, cuando el cliente se ha podido validar como un usuario de la aplicación, recibe una cadena encriptada como respuesta. Esa cadena es el token y sirve para que, en los siguientes accesos, el usuario pueda informar al servidor que ya ha pasado por el proceso de autenticación.

El estándar define la estructura interna que contiene el token, toda la información contenida, se presenta en objetos de tipo JSON que codificados en Base64 e incrustados formando una larga cadena de caracteres firmada digitalmente, lo que hace que dicha información pueda ser verificada y confiable.

Un ejemplo de cómo resulta la apariencia de un token, es el siguiente:


```
accessToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
   eyJpZG1lZGljbyI6MSwidXNlcm5hbWUiOiJrZXlsYSIsInJvbCI6ImFkbWluIiwibm9tYnJlIjoiS2V
   JhcGVsbGlkb3MiOiJDcnV6IFZlcmEiLCJpYXQiOjE2MjQ5MTA2NzMsImV4cCI6MTYyNDkxNDI3M30.
   epXy_pxygKtytF0ZQWboQI32uO41F7jEsmkApU9cx90"
```


Figura 1-6. Ejemplo de token JWT

Como se observa, esta cadena consta de tres partes separadas por puntos, que corresponden a:

- Cabecera: indica el algoritmo y tipo de token.
- Cuerpo (payload): contiene la información del usuario que se hayan incluido convenientemente, el rol que define los privilegios que tiene ese usuario para pedir al servidor ciertas peticiones y también se puede apreciar una fecha de creación iat, y una fecha de expiración exp, ese rango de tiempo es la validez (también configurada en el servidor) que el token estará en vigor.


```
{
  "idmedico": 1,
  "username": "keyla",
  "rol": "admin",
  "nombre": "Keyla",
  "apellidos": "Cruz Vera",
  "iat": 1624910673,
  "exp": 1624914273
}
```


Figura 1-7. Payload del token JWT

- Firma digital: para verificar si el token es válido

Usando un decodificador online, podemos ver la información contenida en el token y ver cómo está conformado.

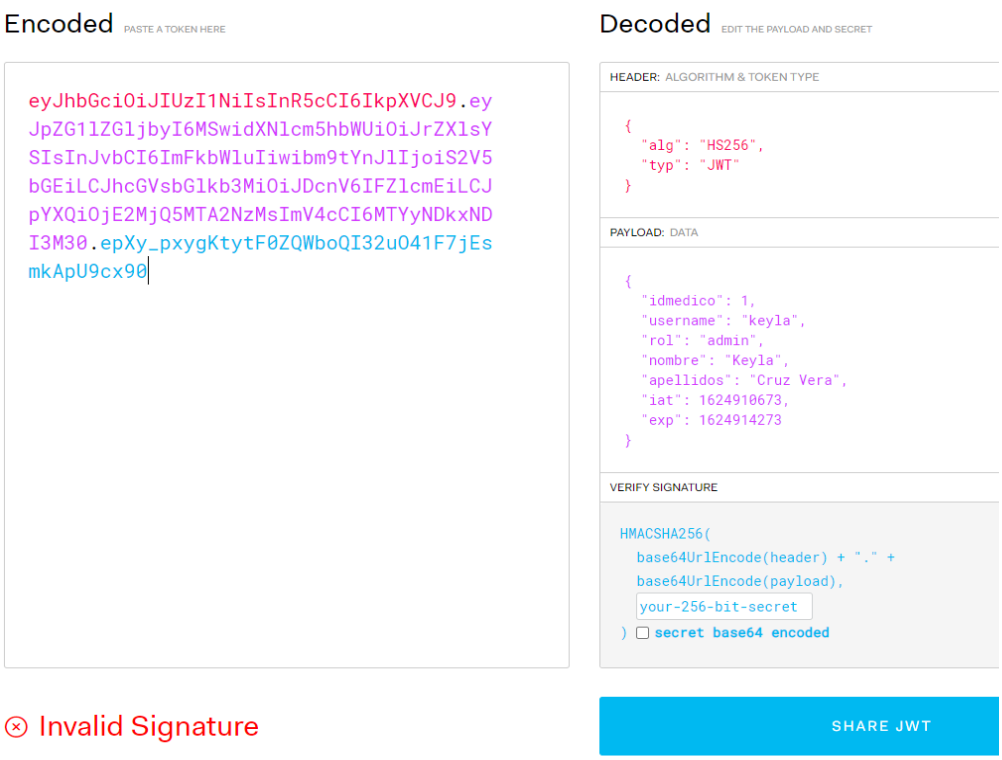


Figura 1-8. Información decodificada contenida en un token JWT

Como podemos observar, la firma se construye de tal forma que podemos verificar que el remitente, es quien dice ser y que el mensaje no se ha modificado nada por el camino.

La firma ha sido construida como HMACSHA256 (Código de autenticación de mensaje cifrado con el algoritmo 256 bits) que contiene la cabecera y el cuerpo codificados en Base64 y un código secreto (establecido en el servidor), necesario para validar el token.

Figura 1-9. Estructura de la firma digital de un token JWT

Por tanto, si se diera la situación de que malintencionadamente se modificara el token para poder inyectar alguna credencial o privilegio, la comprobación de la firma sería incorrecta, el servidor desconfiará del token recibido y denegará la petición recibida

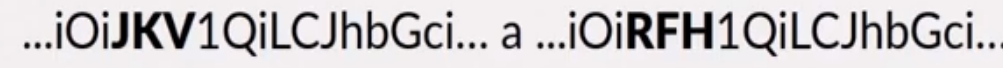


Figura 1-10. Intención de modificar la información de un token JWT

Así, podemos ver que el ciclo de vida, es el siguiente.

1. El usuario accede al sistema introduciendo sus credenciales.
2. Accede al servidor de autenticación.
3. Si las credenciales introducidas corresponden a algún usuario registrado en base de datos, se crea y se le devuelve un token JWT con la información descrita anteriormente y sobretodo firmada digitalmente.
4. A partir de aquí, el usuario podrá hacer todas las peticiones que quiera, siempre y cuando se incluya el token proporcionado en la cabecera de todas las solicitudes que le haga al servidor (el token será válido mientras no haya expirado según su parámetro exp).
5. El servidor comprobará la autenticidad del token y verificará que el token es confiable, es decir que no haya sido maliciosamente manipulado, corroborando la firma digital que contiene un código secreto al que solo el servidor tiene acceso.
6. Si todo es correcto, el servidor enviará la respuesta a la petición del cliente.

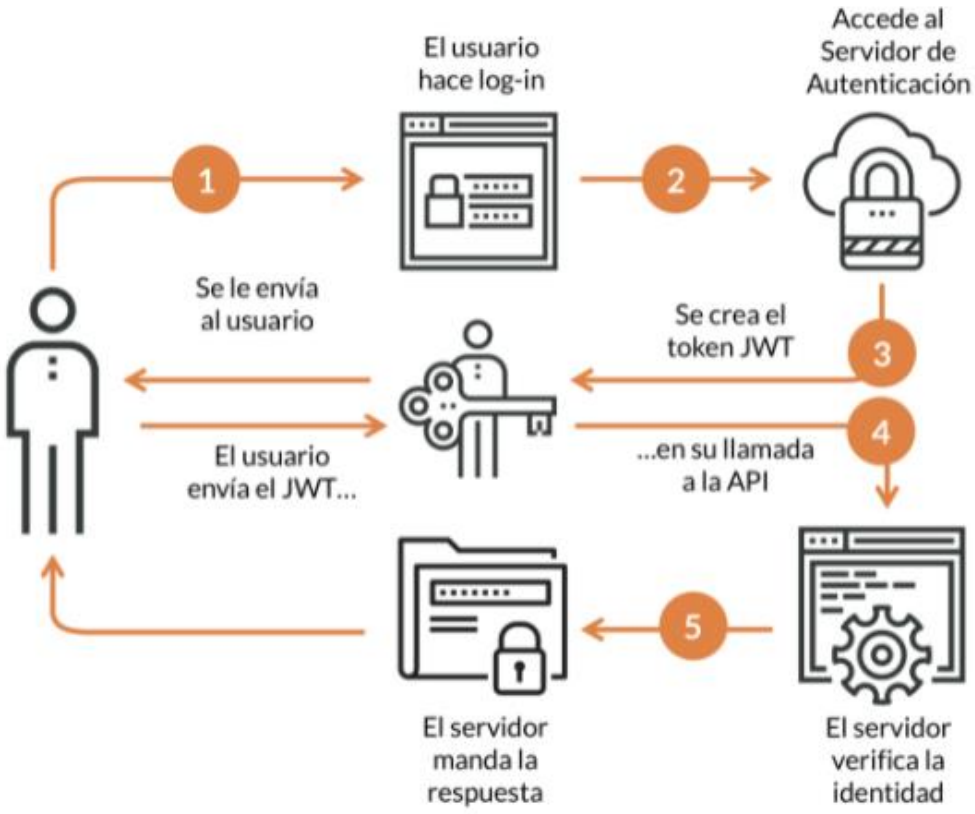


Figura 1-11. Ciclo de vida de un token JWT

1.2.1.6 Firebase

Es una plataforma en la nube adquirida por Google en 2014 y que presta gran variedad de herramientas para **el** desarrollo de aplicaciones web y móvil enfocado principalmente a la integración de sistemas en tiempo real.

Está disponible para distintas plataformas iOS, Android y web.

Figura 1-12. Logo de Firebase de Google

De todos los servicios que presta Firebase, en este proyecto se ha consumido el que cubre el envío de notificaciones.

- **Firebase Cloud Messaging** es una plataforma para el envío de mensajes y notificaciones de forma segura y que está soportado para Android, iOS, y aplicaciones web a varios usuarios en tiempo real y que actualmente puede ser usada de forma gratuita.

En la plataforma de Firebase se crea un proyecto que nos permite descargar ciertas llaves y credenciales para asociar los datos que queremos registrar en el api de Firebase de Google.

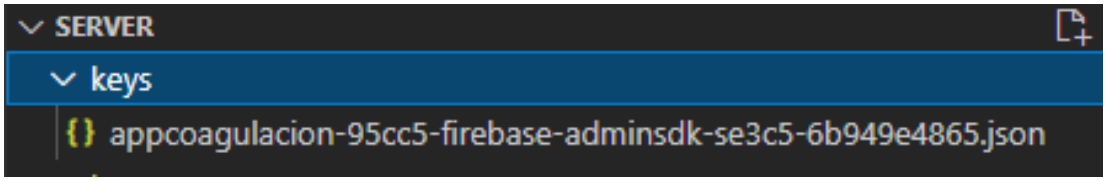


Figura 1-13 . Llave generada en Firebase para asociación de proyecto

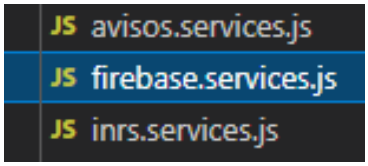


Figura 1-14. Servicio de Firebase integrado en el API

```
import { google } from "googleapis";

const path = "https://fcm.googleapis.com/v1/projects/appcoagulacion-95cc5/messages:send";
const serviceAccount = require("../../keys/appcoagulacion-95cc5-firebase-adminsdk-se3c5-6b949e4865.json");

const scopes = [
    "https://www.googleapis.com/auth/userinfo.email",
    "https://www.googleapis.com/auth/firebase.messaging",
];

const jwtClient = new google.auth.JWT(
    serviceAccount.client_email,
    null,
    serviceAccount.private_key,
    scopes
);
```

Figura 1-15. Uso de la llave para la inicialización de Firebase en el API

```
sendPushNotification: async (token, notification) => {
    let accessToken = "";

    jwtClient.authorize((error, tokens) => {
        if (error) {
            console.log(
                "Error al realizar la solicitud para generar el token de acceso:",
                error
            );
        } else if (tokens.access_token === null) {
            console.log(
                "La cuenta de servicio proporcionada no tiene permiso para generar tokens de acceso"
            );
        } else {
            accessToken = tokens.access_token;
            fetch(path, {
                method: "POST",
                headers: {
                    "Content-Type": "application/json",
                    Authorization: "Bearer " + accessToken,
                },
                body: JSON.stringify({
                    message: {
                        token,
                        notification,
                    },
                }),
            });
        }
    });
},
```

Figura 1-16. Autorización y generación de token para autenticación en el Api de Google

1.2.2 Lado del cliente

La aplicación web, ha sido desarrollada con Angular principalmente para conseguir una aplicación basada en componentes, es decir, un tipo de aplicación Javascript denominada SPA (Single Page Application) y así lograr la máxima velocidad posible en la plataforma web consiguiendo que la experiencia de usuario sea cómoda incluso manejando gran cantidad de datos.

1.2.2.1 Angular

Angular es un framework o entorno de trabajo para aplicaciones web desarrollado en TypeScript, de código abierto, creado y mantenido por Google, cuya finalidad es el desarrollo de aplicaciones web escalables SPA (Single Page Application) basadas en componentes.

Figura 1-17. Logo de Angular

La construcción de aplicaciones de una sola página, sirve para que la interacción y la navegación entre las secciones de esa página se hagan en una sola, de manera que no hay que recargar la página completa con cada uno de esos cambios de sección o con cada petición que se hace al servidor.

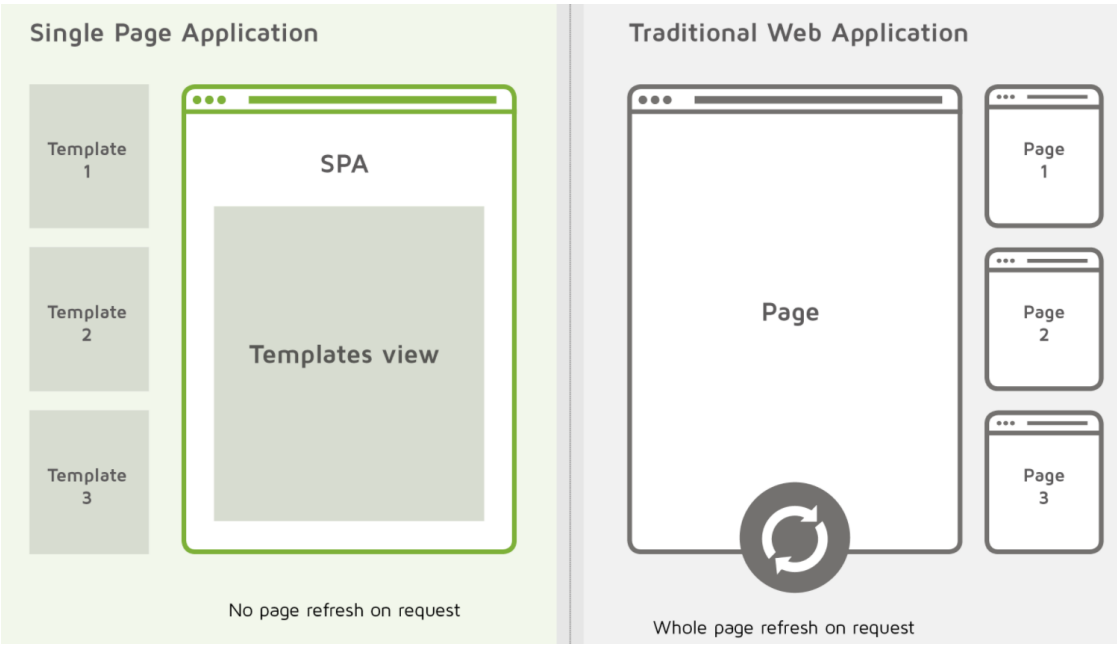


Figura 1-18. Comparación entre aplicaciones de una sola página con las webs tradicionales

El hecho de prescindir de la acción de recargar el navegador con cada cambio o cada evento se debe a que se construye la aplicación basándola en componentes, es decir, en bloques de construcción independientes que componen la aplicación y que son reutilizables.

Un componente incluye:

- plantilla HTML donde va a apariencia de los datos de la clase.
- hoja de estilos Css, Sass o Less que se le aplicarán a la vista HTML del componente si se quiere modificar algo que no deba respetar los estilos generales que se hayan configurado para la aplicación.
- clase donde se desarrolla toda la lógica, atributos de entrada/salida y métodos, y donde define un selector que luego podrá usarse como una etiqueta en la vista HTML de otro componente.

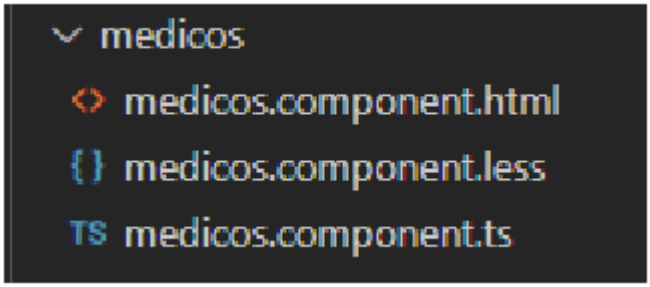
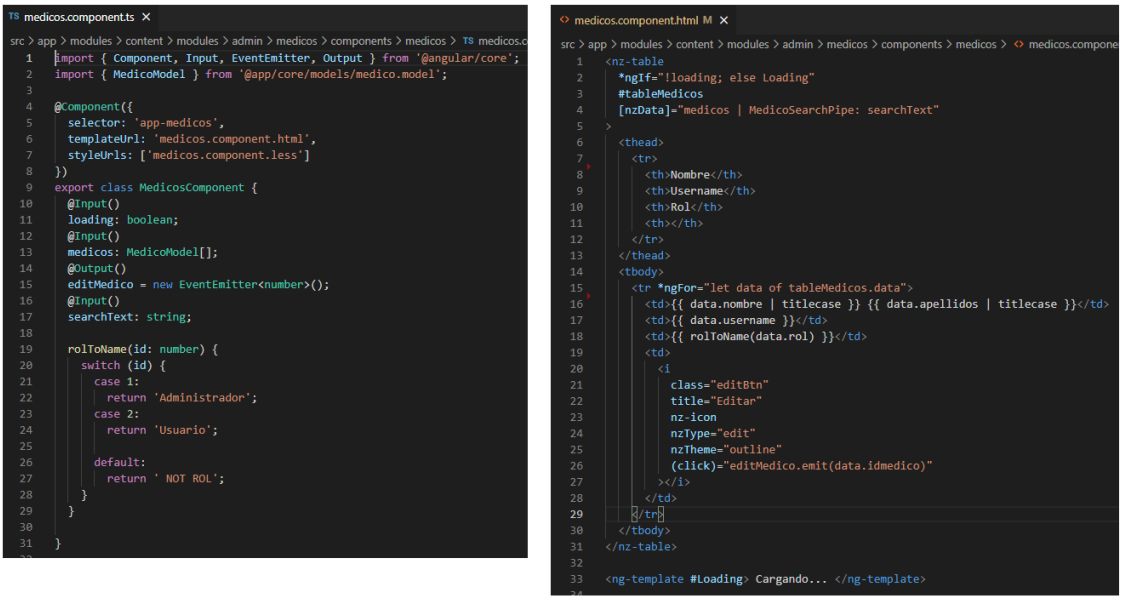


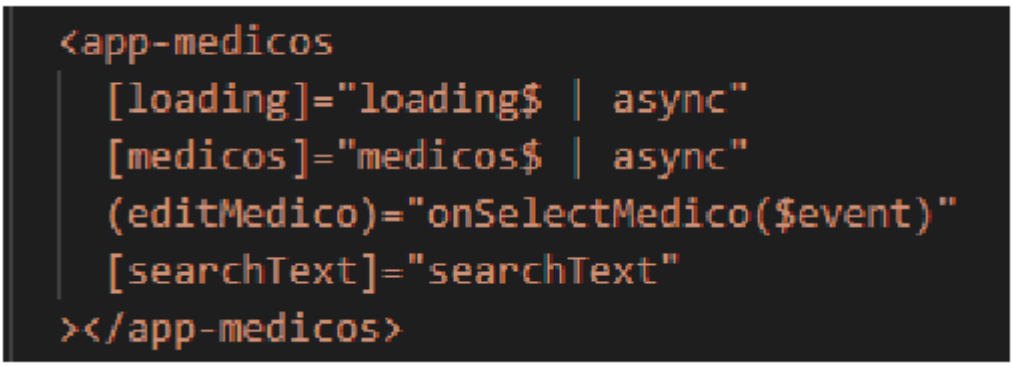
Figura 1-19. Partes de un componente Angular



```ts
// medicos.component.ts
import { Component, Input, EventEmitter, Output } from '@angular/core';
import { MedicoModel } from '@app/core/models/medico.model';

@Component({
  selector: 'app-medicos',
  templateUrl: 'medicos.component.html',
  styleUrls: ['medicos.component.less']
})
export class MedicosComponent {
  @Input()
  loading: boolean;
  @Input()
  medicos: MedicoModel[];
  @Output()
  editMedico = new EventEmitter<number>();
  @Input()
  searchText: string;

  rolToName(id: number) {
    switch (id) {
      case 1:
        return 'Administrador';
      case 2:
        return 'Usuario';

      default:
        return ' NOT ROL';
    }
  }

}
```

```html
<!-- medicos.component.html -->
<nz-table
  *ngIf="!loading; else Loading"
  #tableMedicos
  [nzData]="medicos | MedicoSearchPipe: searchText"
>
  <thead>
    <tr>
      <th>Nombre</th>
      <th>Username</th>
      <th>Rol</th>
      <th></th>
    </tr>
  </thead>
  <tbody>
    <tr *ngFor="let data of tableMedicos.data">
      <td>{{ data.nombre | titlecase }} {{ data.apellidos | titlecase }}</td>
      <td>{{ data.username }}</td>
      <td>{{ rolToName(data.rol) }}</td>
      <td>
        <i
          class="editBtn"
          title="Editar"
          nz-icon
          nzType="edit"
          nzTheme="outline"
          (click)="editMedico.emit(data.idmedico)"
        ></i>
      </td>
    </tr>
  </tbody>
</nz-table>

<ng-template #Loading> Cargando... </ng-template>
```

Figura 1-20. Detalle de la clase y la vista de un componente Angular



```html
<app-medicos
  [loading]="loading$ | async"
  [medicos]="medicos$ | async"
  (editMedico)="onSelectMedico($event)"
  [searchText]="searchText"
></app-medicos>
```

Figura 1-21. Uso de un componente Angular

La capacidad de anidar o incrustar componentes permite conseguir una aplicación web dinámica, asíncrona, reactiva y prácticamente instantánea aun manejando grandes cantidades de datos.

Otra característica destacable en Angular, es la separación completa de la implementación del servidor, es decir, se consigue un cliente completamente independiente que solo se limita a consumir los recursos de un servicio Rest.

El ecosistema de Angular, al ser un entorno desarrollado por Google, tiene gran soporte, una colección de bibliotecas bien integradas que cubren una amplia variedad de características, que incluyen enrutamiento, administración de formularios, comunicación cliente-servidor y una gran comunidad activa en la que se van exponiendo y solucionando cualquier tipo de incidencia.

2 DESCRIPCIÓN DE LA APLICACIÓN

2.1 OBJETIVO Y BREVE EXPLICACIÓN

Crear una aplicación web para el uso de médicos en la que se pueda gestionar a pacientes bajo tratamientos con medicamentos anticoagulantes e interacciones con otros fármacos.

En el Anexo A se adjunta el diseño y distribución que se ha seguido.

La aplicación distingue dos tipos de roles: Administrador y Usuario

En base al rol que el médico tenga asignado, podrá acceder a la totalidad de las opciones del menú si tiene rol Administrador o solo a la parte de gestión de sus pacientes si tiene rol de usuario.

Menú

Submenú Administrador

- añadir o editar la información y los permisos de los médicos que puedan acceder a la plataforma
- acceder a la lista de todos los pacientes y asignarles uno o varios médicos
- administrar datos a nivel general que luego cada médico particularmente (con rol Usuario) podrá asignar a sus pacientes como pueden ser crear o editar:
    - tests con número de preguntas variables, una vez que exista respuesta para ese test, éste dejará de ser editable.
    - avisos o notificaciones predeterminadas.
    - lista de reacciones adversas al medicamento que servirá para que los pacientes comuniquen a su médico si padecen alguna por causa de los tratamientos que tengan en vigor.

Submenú Usuario

- Gestionar solo a los pacientes que el médico tiene asignados, como puede ser:
    - Añadir o editar la información de un paciente, así como darle permiso y generar unas credenciales para que este pueda acceder a través de una aplicación móvil.
    - Asignar o desasignar tests y ver si hay registros de tests respondidos.
    - Añadir o editar las pautas de los tratamientos y ver las reacciones adversas al medicamento que el paciente haya ido comunicando.
    - Insertar una medición del INR (International normalized ratio) y visualizar una gráfica que muestra el historial de las variaciones de los valores tomados.
    - Ver las notificaciones que se le han enviado al paciente a través de la aplicación móvil, así como programar notificaciones automáticas o enviar notificaciones personalizadas adjuntando una imagen.

"INR: prueba de laboratorio que evalúa específicamente la vía extrínseca de la coagulación sanguínea. Se usa para determinar la tendencia de la sangre a coagularse ante la presencia de posibles trastornos de la coagulación, como en la insuficiencia hepática, el déficit de vitamina K o cuando el individuo recibe fármacos anticoagulantes orales dicumarínicos como la Warfarina o el acenocumarol (sintrom®)."

Figura 2-1. Riesgo según rango de valores del INR

2.2 MANUAL DE USUARIO DE LA APLICACIÓN WEB PARA LOS MÉDICOS

2.2.1 Acceso a la plataforma

Se trata de una plataforma con acceso web. Para poder entrar, se abre el navegador web y se accede al siguiente enlace: http://www.appmedicas.es

Aparecerá la pantalla inicial donde se podrá introducir el usuario y la contraseña para que el médico pueda acceder.

Figura 2-2. Pantalla de inicio de sesión de la aplicación web

2.2.2 Cabecera

Figura 2-3. Cabecera de la aplicación web

El botón  sirve para contraer o expandir el menú de la izquierda y así poder aprovechar más el entorno de visualización.

Para poder salir del aplicativo, se puede cerrar la sesión a través del botón  situado en la parte superior derecha de la pantalla.

2.2.3 Menú

El menú, situado en la parte izquierda, consta de dos desplegables a los que se podrá acceder dependiendo del rol que tenga el usuario con el que se ha iniciado sesión.

Si se tiene rol:

- **Administrador**: se podrá acceder al menú completo.
- **Usuario**: solo se podrá ver el panel de usuario.

Figura 2-4. Menú completo de la aplicación web

2.2.3.1 Administrador

El panel de Administrador lo conforman los siguientes módulos:

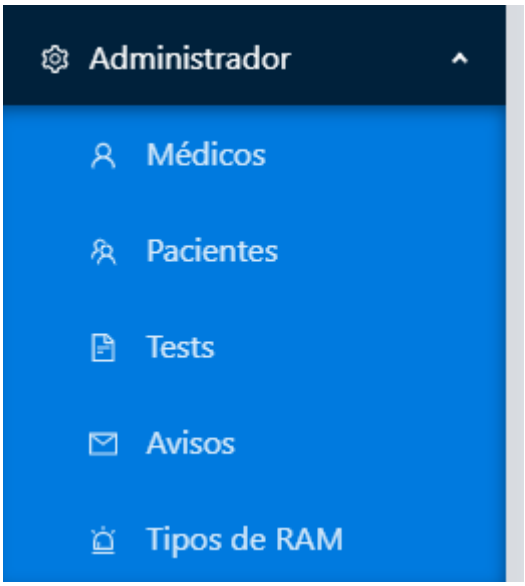


Figura 2-5. Menú de Administrador de la aplicación web

2.2.3.1.1 Médicos

En este apartado, se puede observar una lista con los médicos registrados en el sistema y su información correspondiente, como el nombre completo, el nombre de usuario y el rol que tienen.

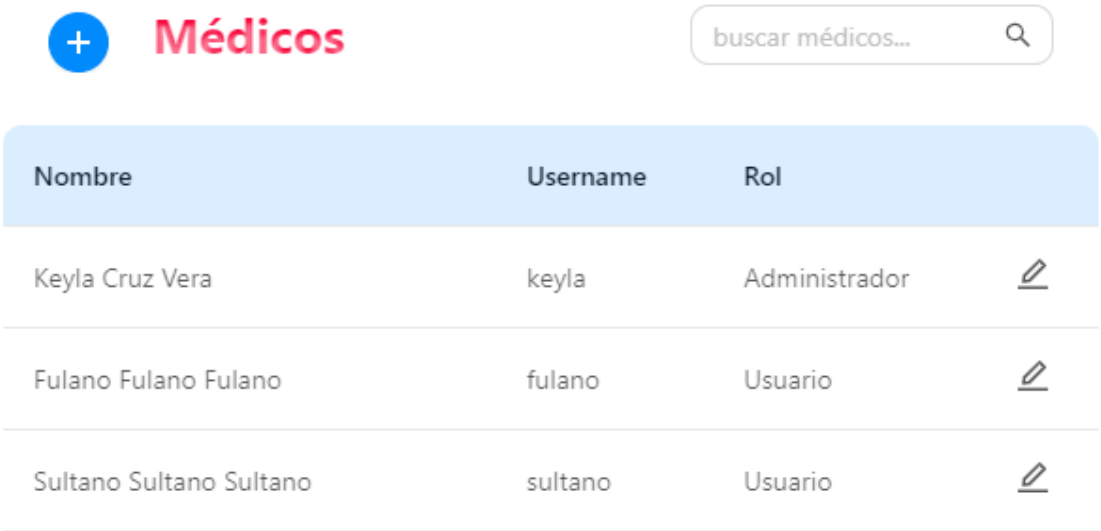


Figura 2-6. Lista de Médicos registrados en el sistema

En la parte superior derecha se ha habilitado un buscador que va filtrando los resultados deseados, incluso si

solo se teclea parte de la palabra.

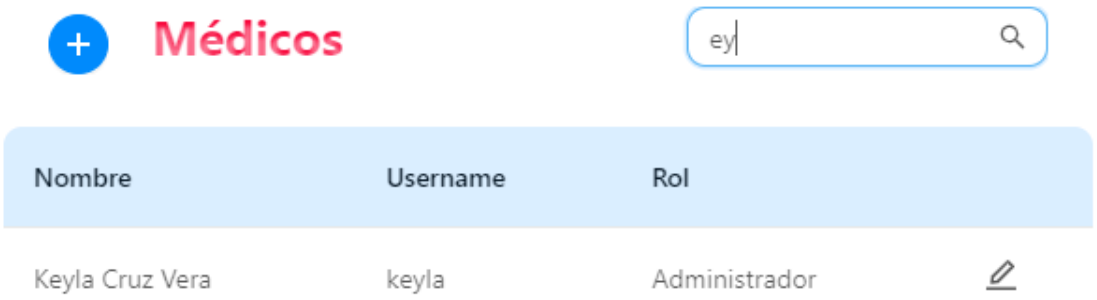


Figura 2-7. Funcionamiento del buscador

Tanto para crear  como para editar  los datos de un usuario, se despliega a la derecha de la pantalla un formulario para que introduzca los parámetros requeridos.

El botón  solo se habilitará cuando la información requerida se haya completado.

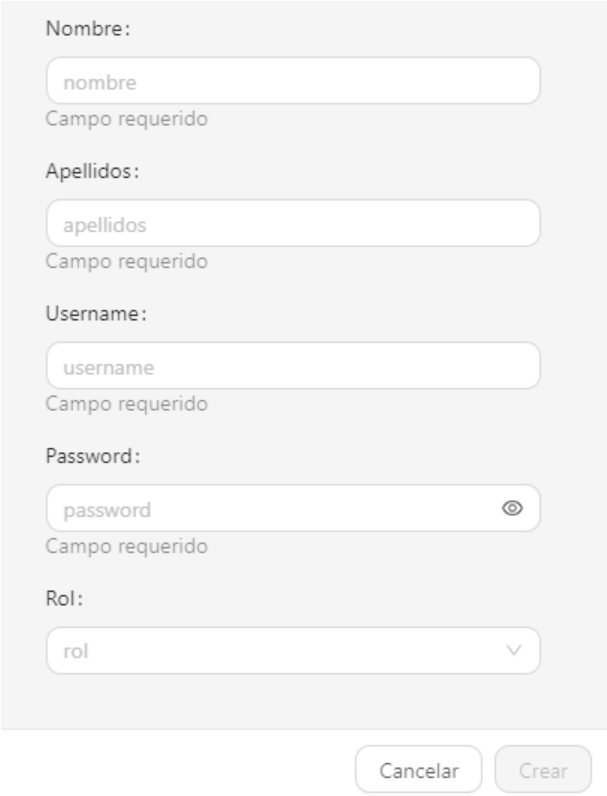


Figura 2-8. Formulario de alta o edición de un médico

Si se quiere borrar un usuario, el botón destinado para ello se encuentra en el desplegable que aparece cuando se pulsa 

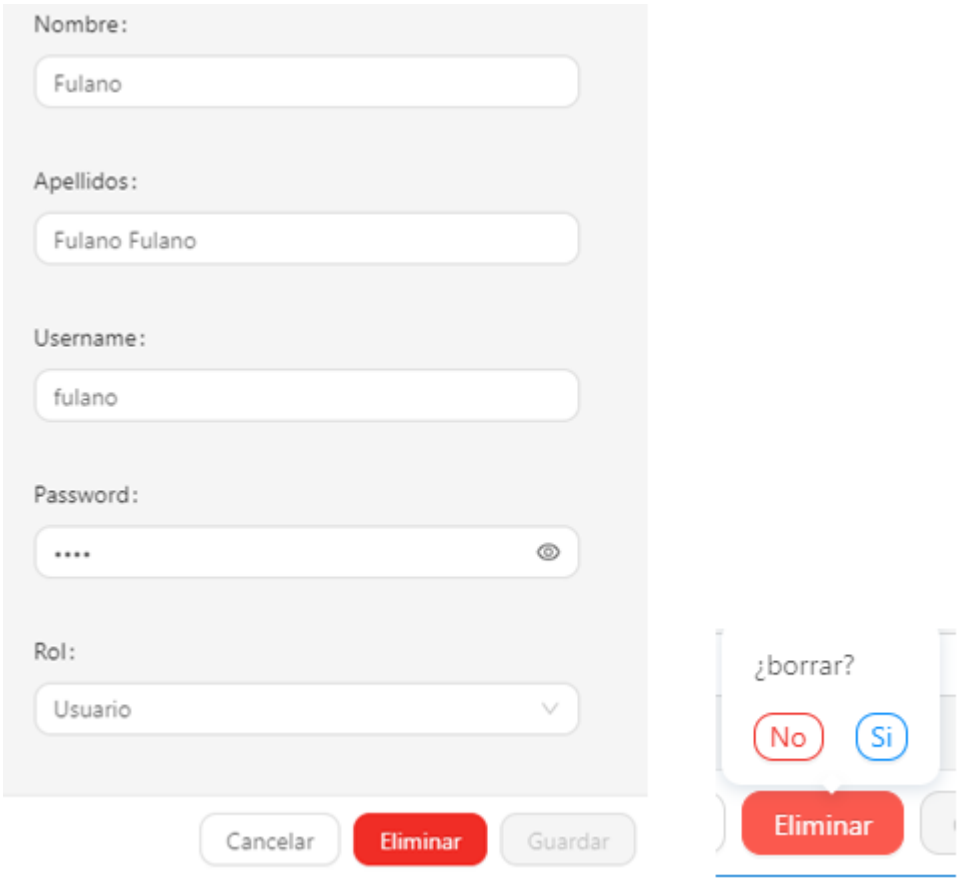


Figura 2-9. Eliminar el registro perteneciente a un médico

2.2.3.1.2 Pacientes

En este apartado, están listados todos los pacientes que estén registrados en el sistema, mostrando su nid (número de identificador) y su nombre completo.

También se ha habilitado el buscador situado en la parte superior derecha.

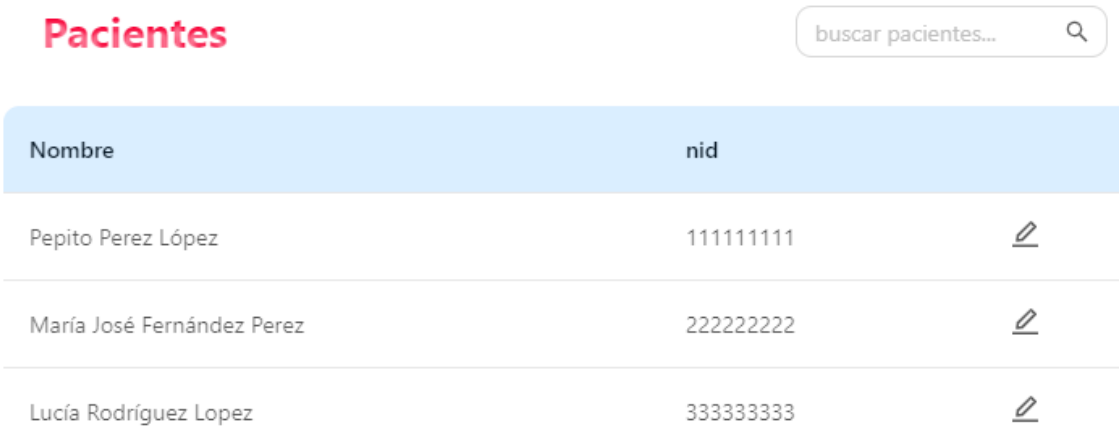


Figura 2-10. Lista de todos los pacientes registrados en el sistema

Si se pulsa  , se despliega a la derecha de su pantalla un formulario para poder asignar a ese paciente uno o varios médicos que hagan su seguimiento.

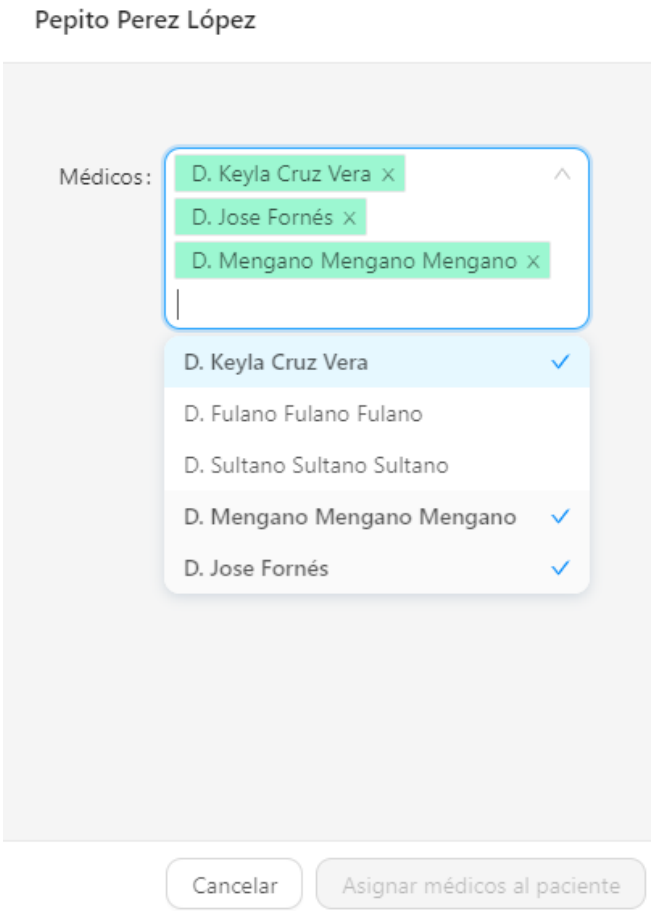


Figura 2-11. Asignación de uno o varios médicos a un paciente

2.2.3.1.3 Tests

En este apartado están listados todos los tests que los médicos administradores pueden ir creando para que luego cada médico, pueda asignarlos a sus pacientes.

Se ha habilitado el buscador situado en la parte superior derecha.

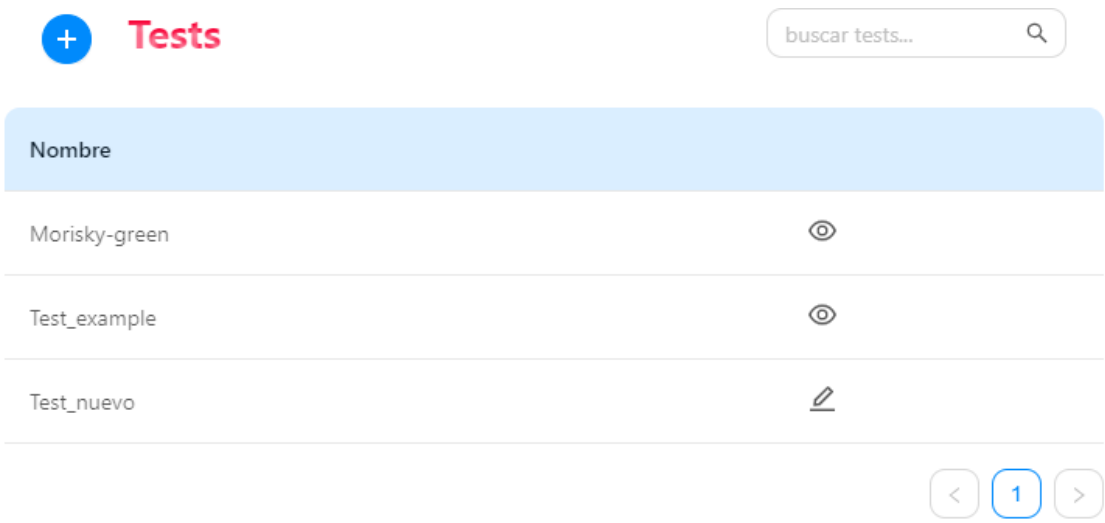
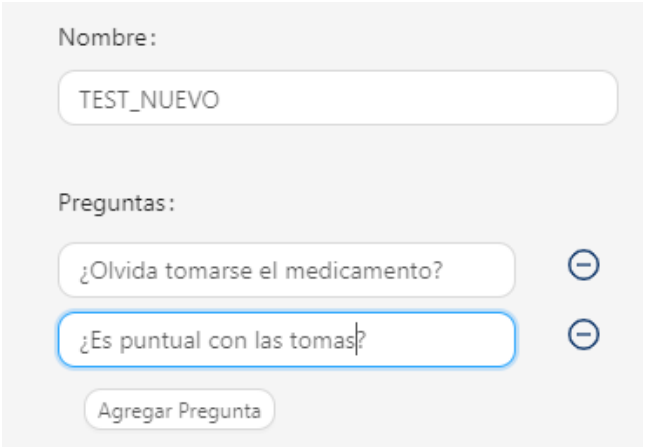


Figura 2-12. Lista de Tests creados por un perfil administrador

Si se pulsa  se desplegará a la derecha de la pantalla un formulario en el que va a rellenar el nombre del test e irá agregando las preguntas oportunas.



A la derecha del listado se pueden observar dos iconos diferentes.

Figura 2-13. Formulario de alta o edición de un test

Si el icono es quiere decir que ese test ya tiene respuestas generadas por los pacientes y por lo tanto no se puede ni modificar ni borrar.

Si el icono es quiere decir que no existen respuestas asociadas y se puede editar o borrar.

Para borrar, en el desplegable que aparece si se pulsa hay un botón para ello.

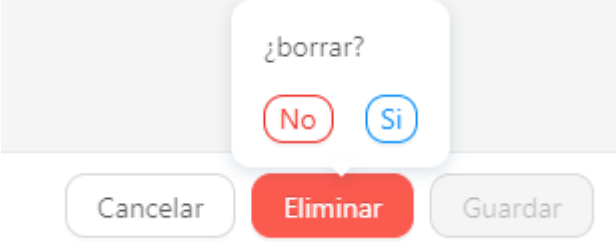


Figura 2-14. Eliminación de un test

2.2.3.1.4 Avisos

En este apartado están listados todos las notificaciones predeterminadas o automáticas que los médicos podrán programar ciertos días a sus pacientes.

Estas notificaciones se usarán como recordatorios de pautas generales y que el paciente recibirá en su terminal móvil los días fijados por su médico.

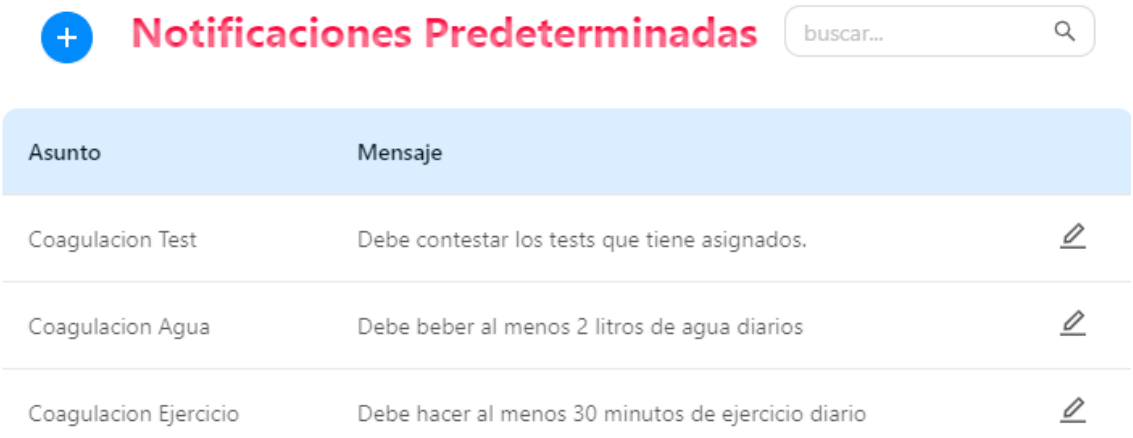


Figura 2-15. Lista de notificaciones predeterminadas creadas por un perfil administrador

Tanto para crear  como para editar  los datos de un aviso o notificación predeterminada, se despliega a la derecha de la pantalla un formulario para que introduzca los parámetros requeridos.

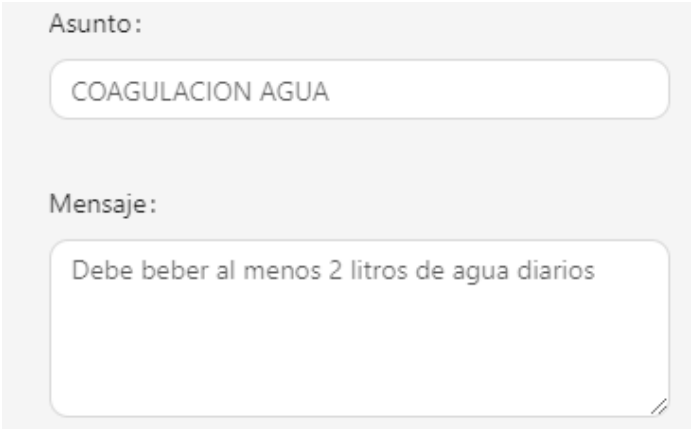


Figura 2-16. Formulario de alta o edición de una notificación predeterminada

Para borrar, en el desplegable que aparece si se pulsa  hay un botón para ello.

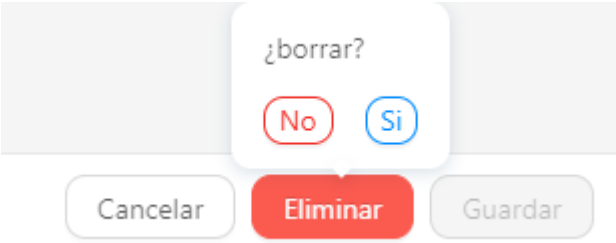


Figura 2-17. Eliminación de una notificación predeterminada

2.2.3.1.5 Tipos de RAM

En este apartado, están listados los tipos de RAM (Reacciones adversas al medicamento), es decir, los efectos secundarios más habituales en los tratamientos con medicamentos anticoagulantes.

Esta lista de efectos es la que los pacientes podrán usar para registrar lo que van sintiendo mientras continúen con sus tratamientos.

Figura 2-18. Lista de tipos de RAM creados por un perfil administrador

Tanto para crear como para editar una RAM o efecto adverso, se despliega a la derecha de la pantalla un formulario para que introduzca los parámetros requeridos.

Para borrar, en el desplegable que aparece si se pulsa hay un botón para ello.

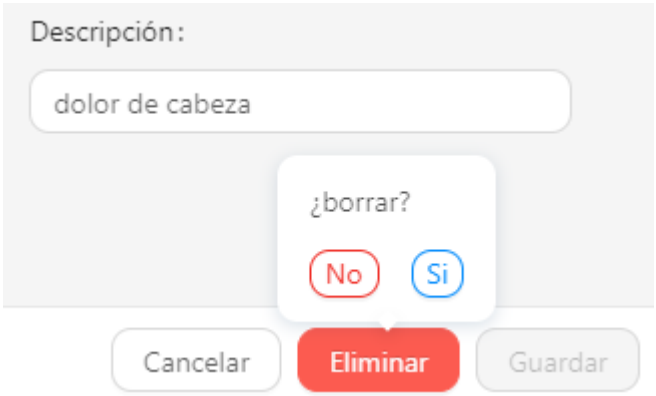


Figura 2-19. Formulario de alta o edición de un tipo de RAM

2.2.3.2 Usuario

El panel de Usuario lo conforma un único módulo:

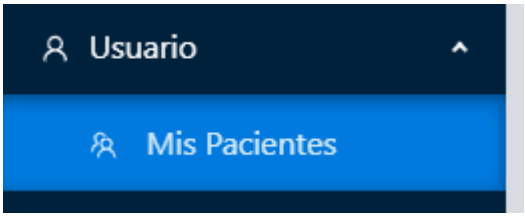


Figura 2-20. Menú para perfil usuario en la aplicación web

2.2.3.2.1 Mis Pacientes

En este apartado, se puede observar un inventario con los pacientes asignados al médico que ha iniciado sesión.

Como se observa cada registro o instancia al paciente tiene cierta información correspondiente como el nombre completo y el nid (número de identificador).

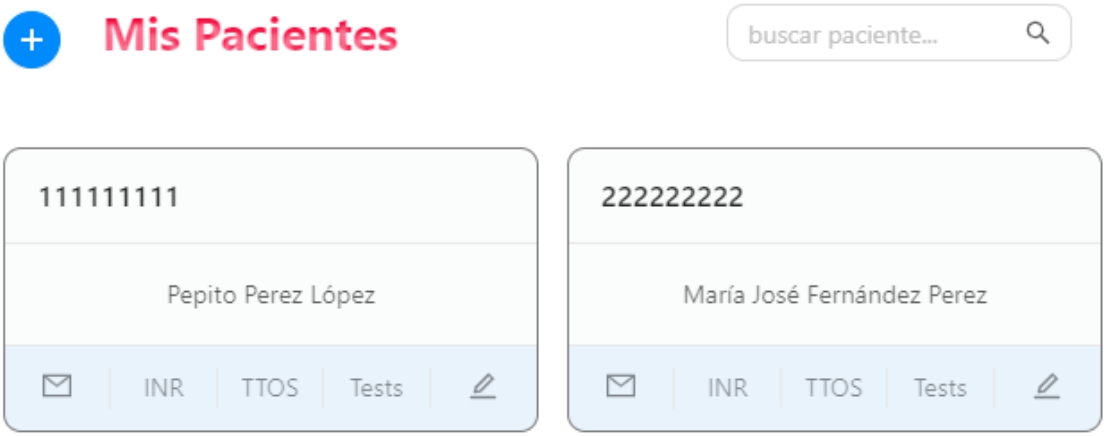


Figura 2-21. Lista de pacientes del médico

Además, se proporciona una serie de opciones para su gestión, que a continuación se detallará:

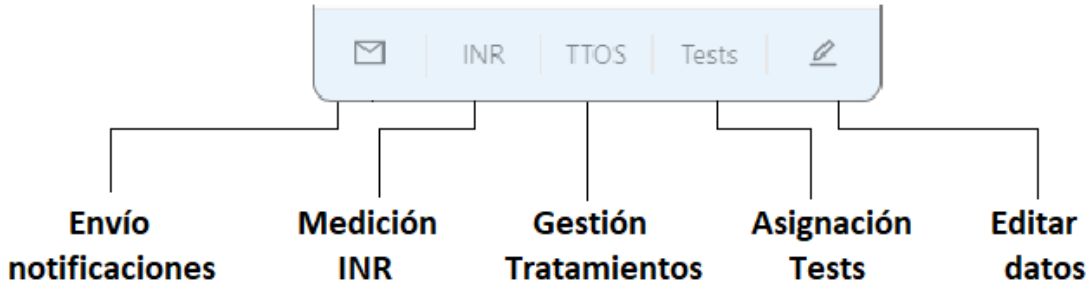


Figura 2-22. Opciones para gestionar a un paciente

2.2.3.2.1.1 Creación y edición de los datos de un paciente

Tanto para crear  como para editar  los datos de un paciente, se despliega a la derecha de la pantalla un formulario para que introduzca los parámetros requeridos.

Entre esos parámetros se puede habilitar el acceso y las credenciales para que el paciente pueda hacer uso de la aplicación móvil (tema de otro proyecto TFG).

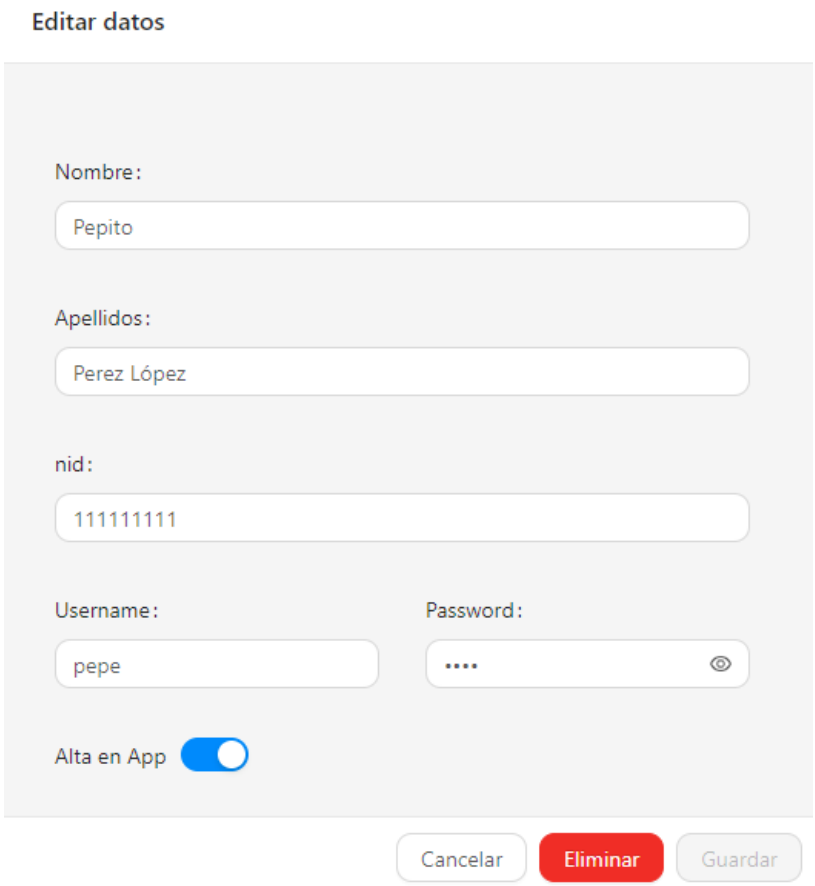


Figura 2-23. Formulario de alta o edición de un paciente

Para borrar, en el desplegable que aparece si se pulsa  hay un botón para ello.

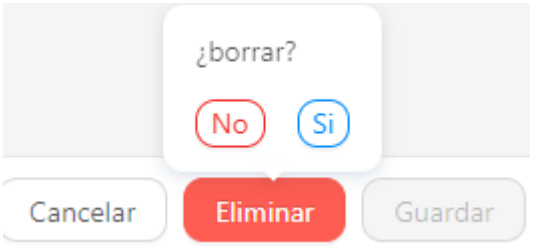


Figura 2-24. Eliminación de un paciente

2.2.3.2.1.2 Asignación de tests y visualización de los tests respondidos

Si se pulsa en Tests aparece a la derecha de la pantalla un desplegable con dos apartados.

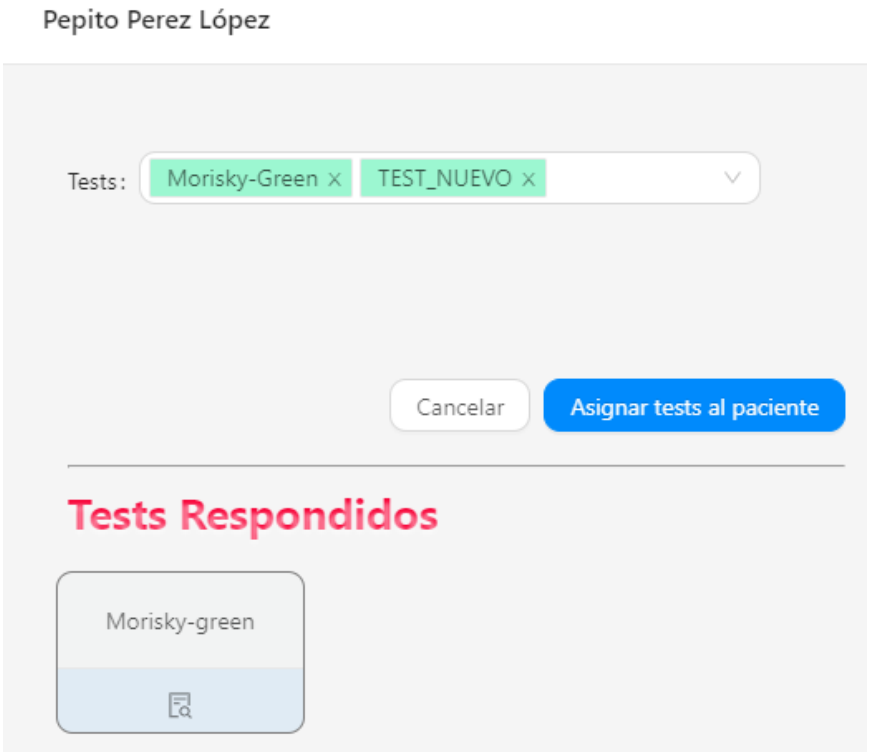


Figura 2-25. Panel de dos módulos relacionados a los tests de un paciente

El primero sirve para que el médico pueda hacer la asignación al paciente de uno o varios de los tests (que se han ido creando en el módulo de tests de administrador visto anteriormente).

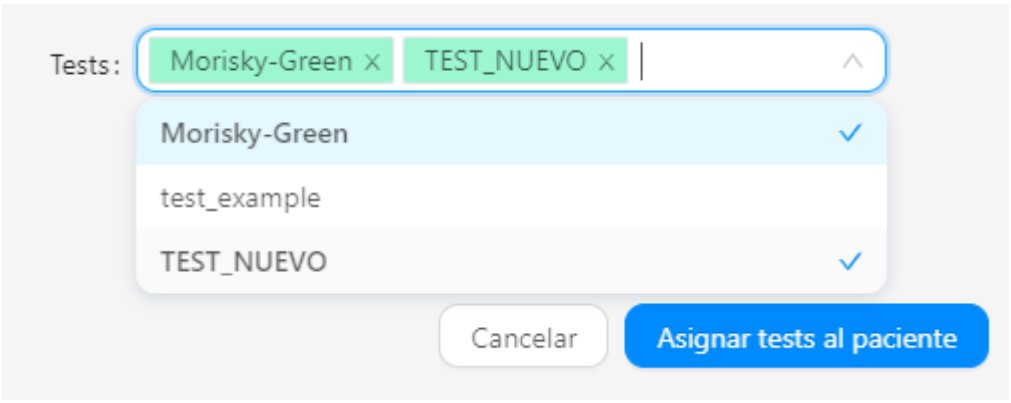


Figura 2-26. Asignación de uno o varios tests a un paciente

El segundo apartado sirve para que el médico pueda visualizar los tests respondidos y las fechas de los mismos.

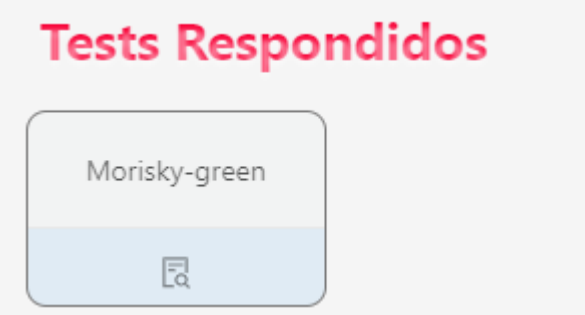


Figura 2-27. Lista de tests que un paciente ha respondido

Si se pulsa en  se abrirá un segundo desplegable con el historial de respuestas indicando la fecha en la que se han ido realizando.

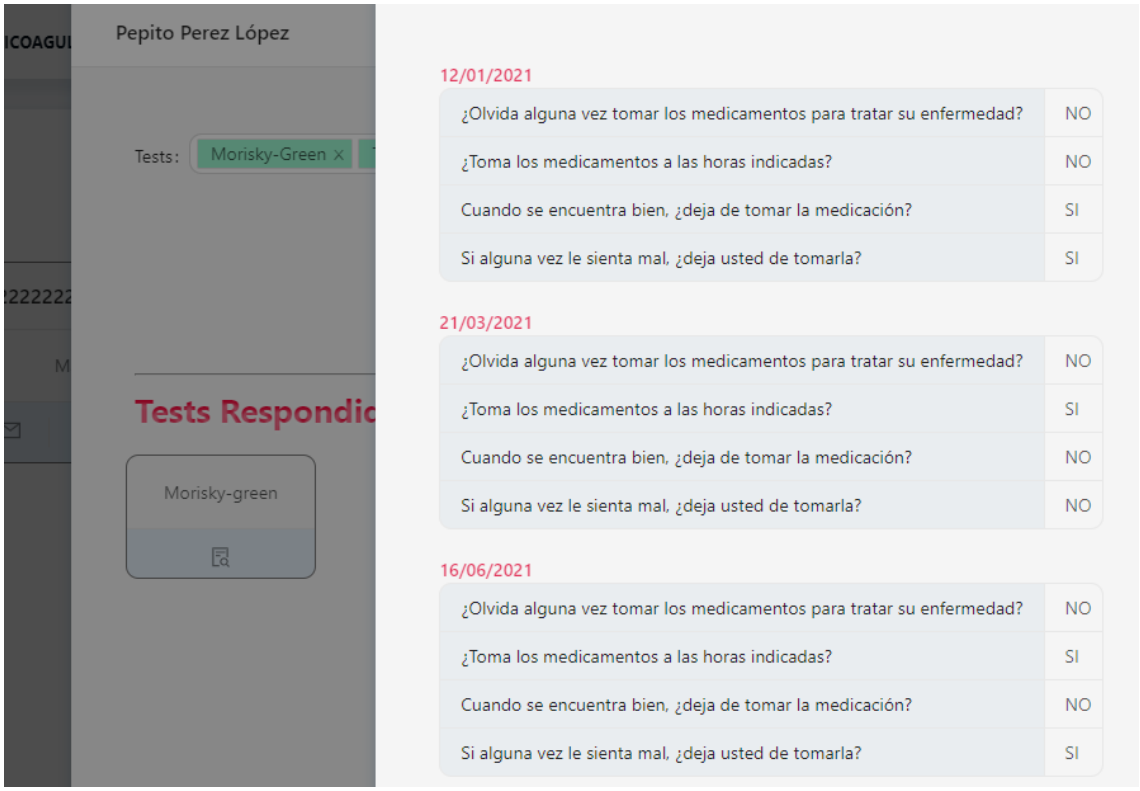
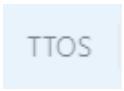


Figura 2-28. Historial de tests respondidos de un paciente

2.2.3.2.1.3 Gestión de los tratamientos del paciente

Si se pulsa en 

se va a redirigir a una pantalla donde se puede ver listados los tratamientos del paciente.

Se muestra el nombre del medicamento, el periodo del tratamiento y la dosis.

En la parte derecha se puede ver que se ha habilitado un buscador.

Para volver a la lista de pacientes, se debe pulsar 

Figura 2-29. Lista de tratamientos de un paciente

Tanto para crear 

como para editar 
los datos del tratamiento de un paciente, se despliega a la derecha de la pantalla un formulario para que introduzca los parámetros requeridos.

Figura 2-30. Formulario de alta o edición de un tratamiento para un paciente

Para borrar, en el desplegable que aparece si se pulsa  hay un botón para ello

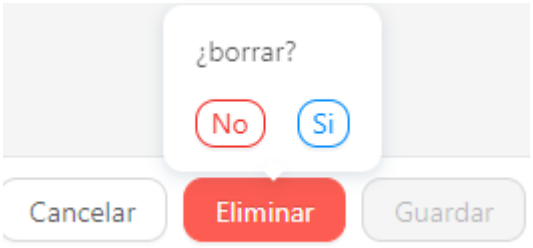


Figura 2-31. Eliminación de un tratamiento de un paciente

Si se pulsa  se va a redirigir a una pantalla donde se puede ver listados las reacciones adversas a ese medicamento (RAM) que el paciente ha ido experimentando durante el tratamiento y que ha ido registrando desde la aplicación móvil.

El paciente elige una RAM de aquellas que se hayan creado en el apartado "Tipos de RAM" en el módulo Administrador.

Para volver a la lista de los tratamientos de ese paciente, se debe pulsar

Nota: En este apartado se han dejado habilitadas las funciones para crear + Insertar RAM editar y borrar Eliminar por si en algún momento el médico tuviera que usarlas. Pero la finalidad es que en este módulo el médico únicamente pueda visualizar los efectos secundarios registrados por el paciente (desde la aplicación móvil).

Figura 2-32. Lista de RAM asociados a un tratamiento de un paciente

2.2.3.2.1.4 Medición del INR

"INR: prueba de laboratorio que evalúa específicamente la vía extrínseca de la coagulación sanguínea. Se usa para determinar la tendencia de la sangre a coagularse ante la presencia de posibles trastornos de la coagulación, como en la insuficiencia hepática, el déficit de vitamina K o cuando el individuo recibe fármacos anticoagulantes orales dicumarínicos como la Warfarina o el acenocumarol (sintrom®)."

En este apartado están listados los valores que van resultando de las pruebas de laboratorio realizadas al paciente.

Figura 2-33. Lista de INR correspondientes a las pruebas realizadas a un paciente

Tanto para crear  como para editar  los datos de una prueba INR del paciente, se despliega a la derecha de la pantalla un formulario para que introduzca los parámetros requeridos, como el valor y la fecha de la prueba.

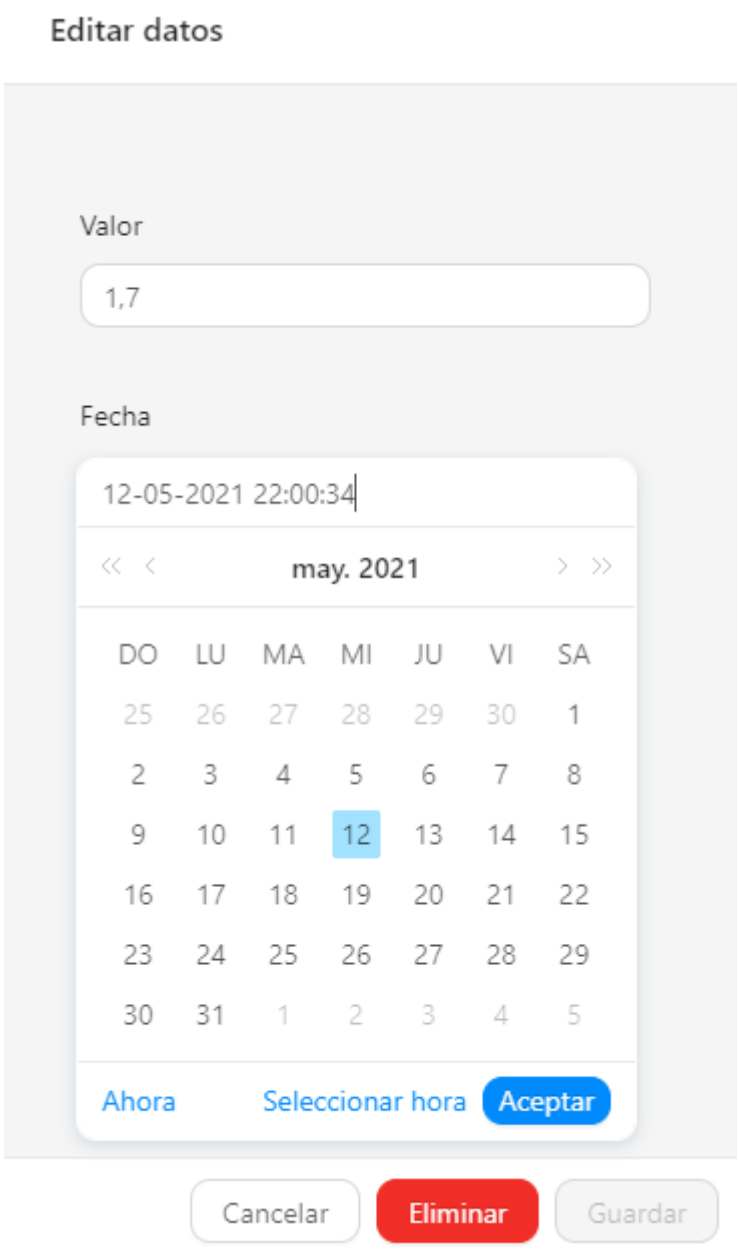


Figura 2-34. Formulario de registro o edición de un valor INR de un paciente

Si se pulsa 

se despliega a la derecha de la pantalla una gráfica que muestra los valores del historial INR del paciente donde se puede apreciar la evolución que ha tenido.

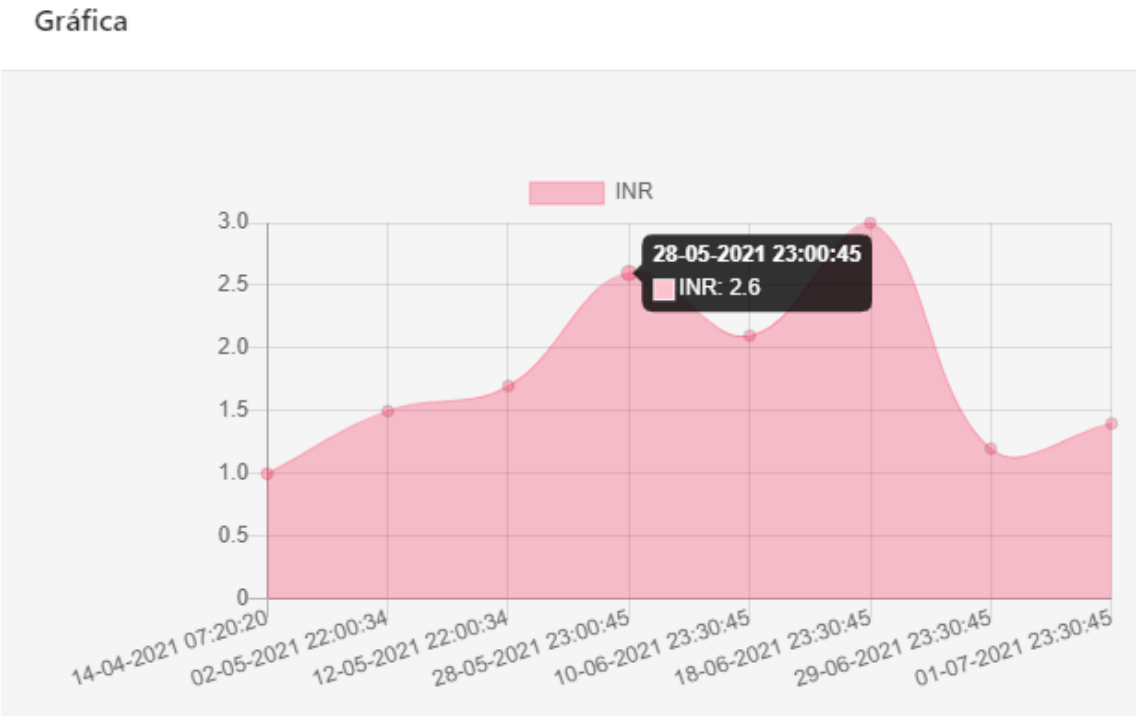


Figura 2-35. Gráfica INR-fecha de un paciente

Para volver a la lista de pacientes, se debe pulsar 

2.2.3.2.1.5 Envío de notificaciones

Si se pulsa  se va a redirigir a una pantalla donde se puede ver una lista con las notificaciones que se le han enviado a un paciente, se listan tanto las notificaciones que se han enviado automáticamente como las que se le ha enviado de forma personalizada.

Figura 2-36. Lista de notificaciones enviadas a un paciente

Para enviar una notificación personalizada, se debe pulsar 

se despliega a la derecha de la pantalla un formulario que deber rellenar con un asunto, el mensaje a enviar y opcionalmente puede adjuntar una imagen.

Figura 2-37. Envío de notificación personalizada a un paciente

Si se pulsa sobre  se abrirá el explorador de archivos donde se podrá elegir la imagen a enviar.

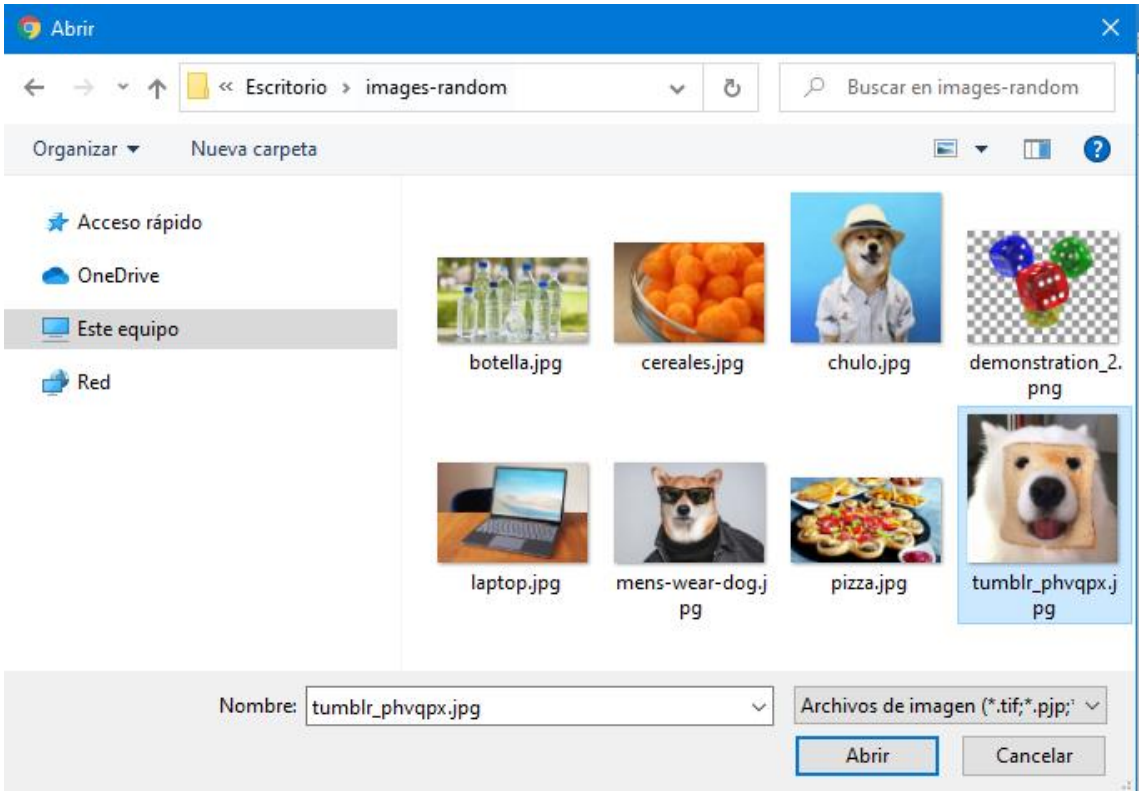
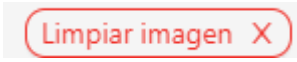


Figura 2-38. Explorador de archivos para seleccionar imagen en la notificación

Se podrá previsualizar la imagen que se va a enviar.

También se dispone de un botón 

para eliminar la imagen seleccionada y poder enviar la notificación sin imagen si se desea.

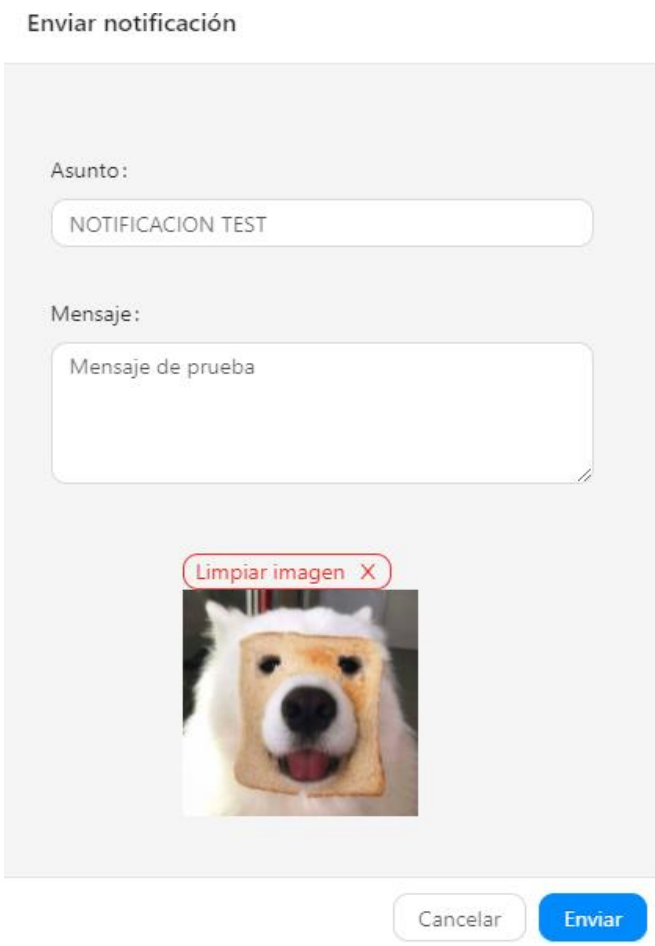


Figura 2-39. Ejemplo de envío de notificación con imagen

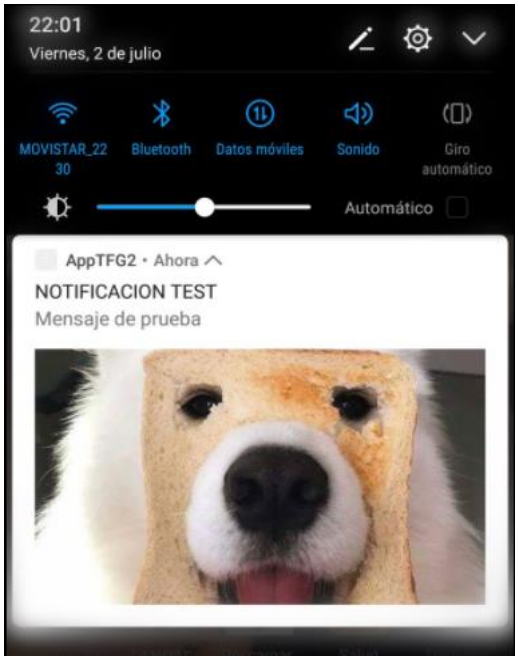


Figura 2-40. Ejemplo de recepción de notificación con imagen en terminal móvil

Figura 2-41. Ejemplo de envío de notificación sin imagen

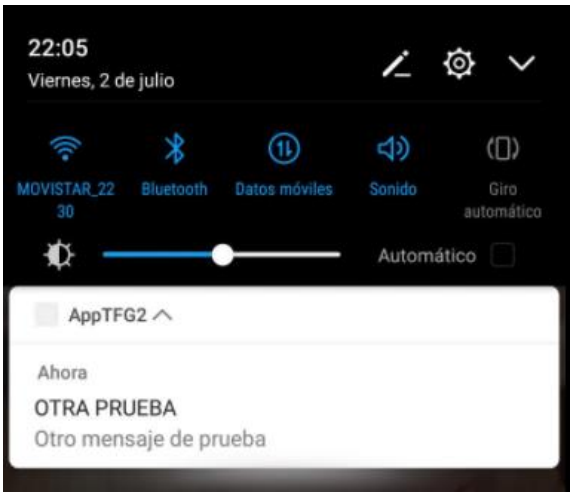
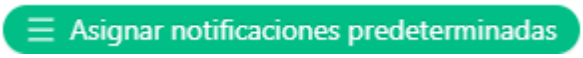


Figura 2-42. Ejemplo de recepción de notificación sin imagen en terminal móvil

Para asignar y programar el envío de notificaciones automáticas, se debe pulsar 

se despliega a la derecha de la pantalla una lista con todos los avisos creados en el apartado “Avisos” del módulo de menú Administrador.

Si se elige una notificación  aparece a la derecha de la línea seleccionada un botón  para determinar los días que va a ser enviada.

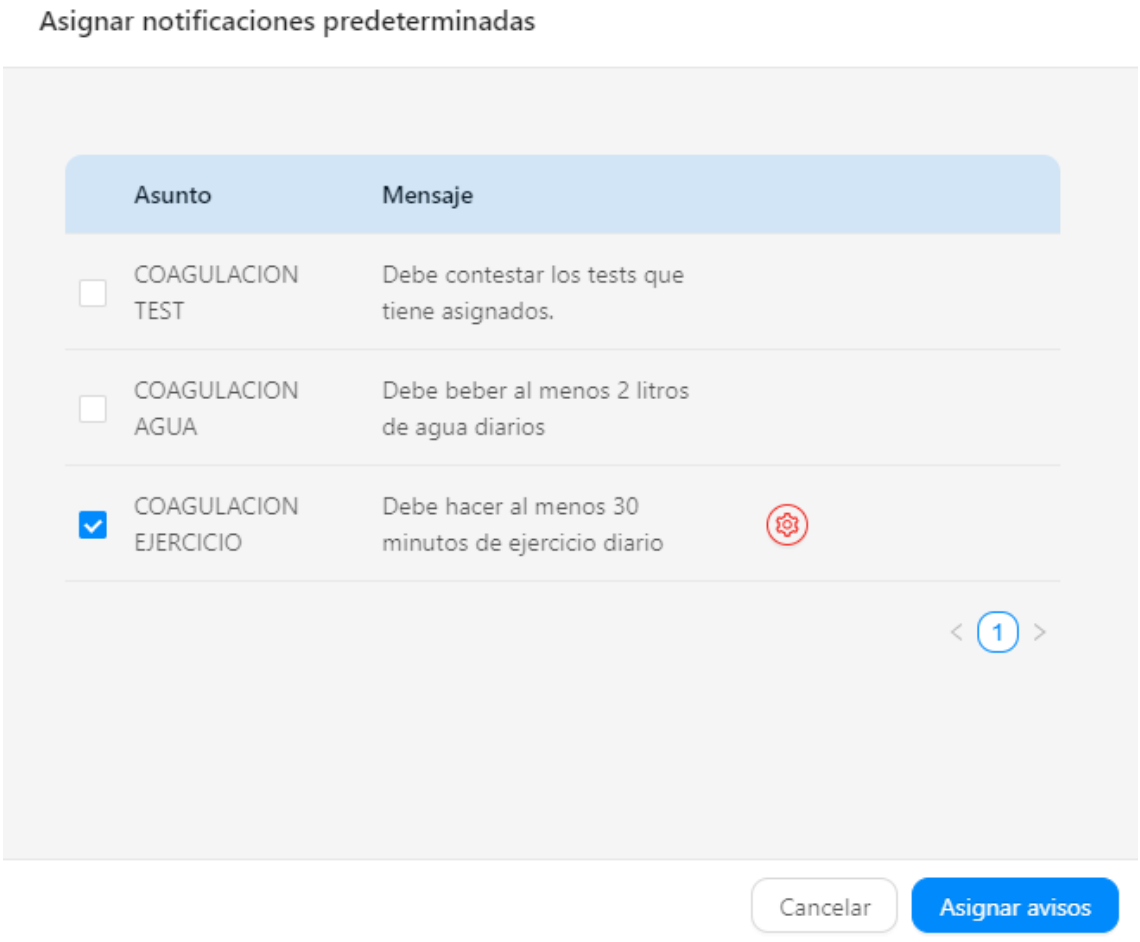


Figura 2-43. Lista de notificaciones predeterminadas para programar el envío a un paciente

Si se pulsa  aparecerá un formulario flotante, para poder elegir los días.

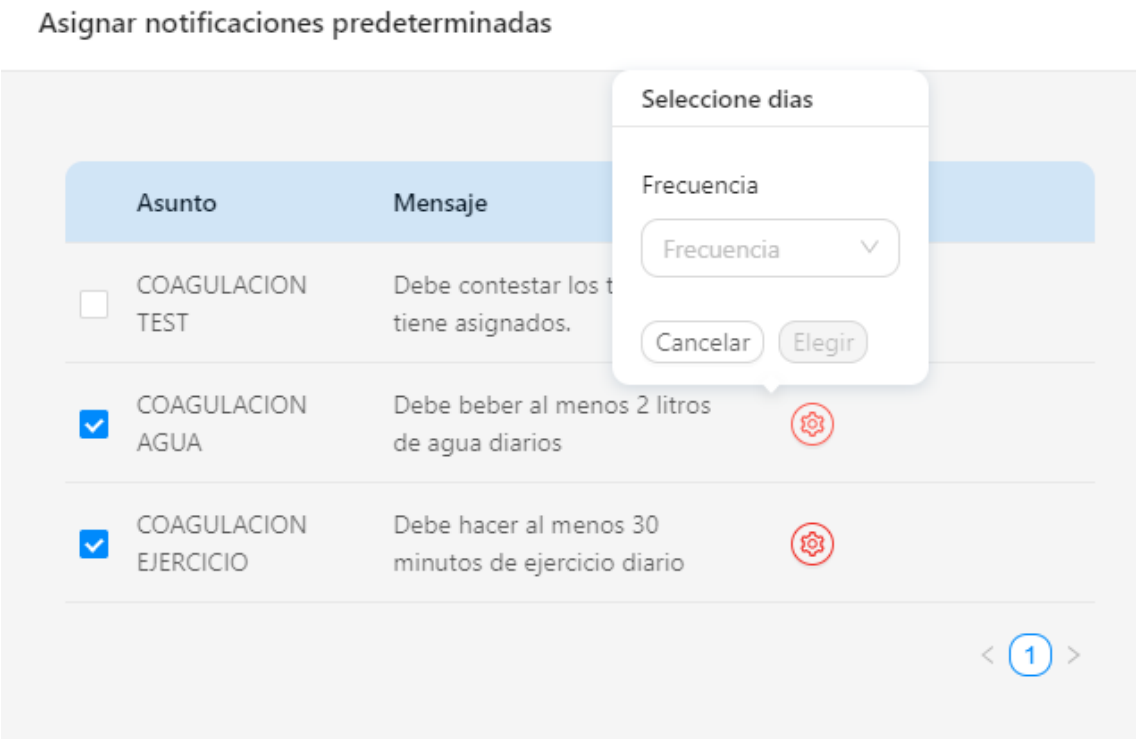
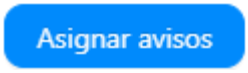


Figura 2-44. Programación de una notificación predeterminada para el envío a un paciente

Figura 2-45. Selección de los días para el envío automático a un paciente

Una vez seleccionados, se debe pulsar Elegir

Se debe repetir la misma secuencia de acciones para todas las notificaciones que se vayan a programar.

Por último, se debe pulsar en 

para completar la programación.

3 PLIEGO DE CONDICIONES

3.1 CONFIGURACIÓN DE LA MÁQUINA (CENTOS7 X64)

Una vez que se me facilitaron las credencias para poder acceder al servidor mediante el protocolo SSH (Secure Shell) como **root**.

Se instalaron las siguientes herramientas detalladas paso a paso:

Nota: Antes de hacer esta instalación de Node.js, debemos haber instalado las herramientas de desarrollador para Centos 7.

```
sudo yum -y install curl
sudo yum -y groupinstall "Development Tools"
```

Instalación de Bash (Bourne-again *shell*)

1. Se debe descargar el archivo tar de Bash 5.0

```
curl -O https://ftp.gnu.org/gnu/bash/bash-5.0.tar.gz
```

2. Se debe extraer el archivo descargado

```
tar xvf bash-5.0.tar.gz
```

3. Para configurarlo

```
cd bash-5.0
./configure
```

4. No se debe apreciar ningún error en la en la salida de ejecución del comando

```
.....
config.status: creating Makefile
config.status: creating builtins/Makefile
config.status: creating lib/readline/Makefile
config.status: creating lib/glob/Makefile
config.status: creating lib/intl/Makefile
config.status: creating lib/malloc/Makefile
config.status: creating lib/sh/Makefile
config.status: creating lib/termcap/Makefile
config.status: creating lib/tilde/Makefile
config.status: creating doc/Makefile
config.status: creating support/Makefile
config.status: creating po/Makefile.in
config.status: creating examples/loadables/Makefile
config.status: creating examples/loadables/Makefile.inc
config.status: creating examples/loadables/perl/Makefile
config.status: creating support/bash.pc
config.status: creating support/bashbug.sh
config.status: creating config.h
config.status: executing default-1 commands
config.status: creating po/POTFILES
config.status: creating po/Makefile
config.status: executing default commands
```

5. Se procede a instalar Bash 5.0

```
make

sudo make install
```

3.1.1 Instalación Node.js y su gestor de paquetes, NPM

1. Se debe descargar el repositorio de NodeSource. Este comando usará primero la herramienta curl para descargar el script y luego ejecutarlo usando bash.

```
[root@appmedicas ~]# curl -sL https://rpm.nodesource.com/setup_12.x | bash -
```

2. Se puede comprobar si el nuevo repositorio se ha agregado correctamente

```
[root@appmedicas ~]# ls -la /etc/yum.repos.d/|grep nodesource
```

3. Se procede a la instalación

```
[root@appmedicas ~]# sudo yum list available nodejs
```

4. Al finalizar la instalación, se verá el mensaje **Complete!**, lo que significa que Node.js y su gestor de paquetes NPM, se han instalado correctamente en el servidor.

3.1.2 Instalación MySQL

1. Se debe actualizar el sistema

```
[root@appmedicas ~]# sudo yum update
```

2. Se debe descargar el repositorio de MySQL

```
[root@appmedicas ~]# sudo wget https://dev.mysql.com/get/mysql80-community-release-el7-3.noarch.rpm
```

3. Finalizada la descarga, se visualizará un mensaje de confirmación de que se guardó el archivo **.rpm,** hay que ejecutarlo para preparar el repositorio y así poder instalar paquetes**.**

```
[root@appmedicas ~]# sudo rpm -Uvh mysql80-community-release-el7-3.noarch.rpm
```

4. Finalizada la ejecución, se instala MySQL.

```
[root@appmedicas ~]# sudo yum install mysql-server
```

5. El script va a devolver una lista de paquetes y pedirá confirmación para descargarlos e instalarlos. Hay que escribir **y** (indica "yes") y pulsar **ENTER** para aceptar cada solicitud.
   Al finalizar la instalación, se verá el mensaje **Complete!**, lo que significa que MySQL se ha instalado correctamente en el servidor.

6. MySQL no se iniciará automáticamente inmediatamente después de la instalación. Por lo tanto, hay que iniciarlo manualmente.

```
[root@appmedicas ~]# sudo systemctl start mysqld
```

7. No se obtendrá respuesta una vez que MySQL se inicie, así que para verificar si está funcionando correctamente, se puede usar

```
[root@appmedicas ~]# sudo systemctl status mysqld
```

8. Aparecerá la información sobre el proceso de MySQL. Si está activo y ejecutándose, se ha instalado e iniciado correctamente MySQL en el servidor.

```
[root@appmedicas ~]# sudo systemctl status mysqld
● mysqld.service - MySQL Server
   Loaded: loaded (/usr/lib/systemd/system/mysqld.service; enabled; vendor preset: disabled)
   Active: active (running) since Mon 2021-03-29 14:09:04 EDT; 3 months 4 days ago
     Docs: man:mysqld(8)
           http://dev.mysql.com/doc/refman/en/using-systemd.html
 Main PID: 3505 (mysqld)
   Status: "Server is operational"
   CGroup: /system.slice/mysqld.service
           └─3505 /usr/sbin/mysqld
```

3.1.3 Instalación de PM2

PM2 (Process Manager 2) es un administrador de procesos, de nivel de producción, de código abierto, avanzado, eficiente y multiplataforma para **Node.js** con un equilibrador de carga incorporado que se puede instalar mediante NPM y que su función es ejecutar un demonio en el sistema operativo del servidor que permita administrar múltiples instancias de node.js dentro del sistema.

```
[root@appmedicas ~]# sudo npm install -g pm2
```

3.2 DESPLIEGUE DEL PROYECTO EN PRODUCCIÓN

A lo largo del desarrollo del proyecto he hecho uso de la herramienta para el control de versiones gitLab.

Figura 3-1. Logo de GitLab

En mi caso, para clonar o descargar el proyecto desde mi repositorio:

1. Se crea una llave para asociar la máquina con el proyecto
   ```
   [root@appmedicas ~]# ssh-keygen
   ```
   ```
   [root@appmedicas ~]# cat ~/.ssh/id_rsa.pub
   ```
2. Se debe agregar esa llave en el perfil de gitLab -> edit profile -> SSH Keys
3. Una vez asociada, ya se puede clonar el proyecto
   git clone <repositorio>

En otro caso, si se tiene el proyecto en un archivo comprimido .zip, basta con descomprimirlo en la máquina.

Se ha creado una carpeta /coagulación que contiene:

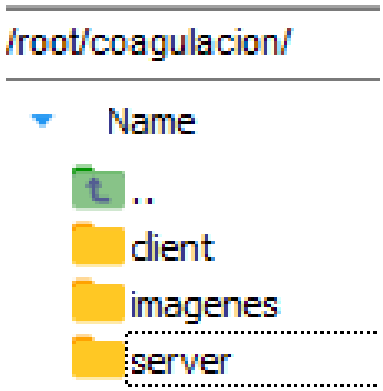


Figura 3-2. Contenido del directorio del proyecto en servidor de producción

/imágenes -> carpeta donde se van guardando las imágenes que se envían con las notificaciones a los pacientes.

Paso a paso:

1. Se entra en la carpeta server
```
[root@appmedicas ~]# cd coagulacion/server/
```
2. Se instalan las dependencias
```
[root@appmedicas server]# npm install
```
3. Se puede consultar en package.json del proyecto donde como se compila el proyecto.
```
[root@appmedicas server]# npm run build
```
4. Una vez compilado, tenemos la carpeta dist donde dentro hay que crear el directorio /public/client-coagulacion
```
/root/coagulacion/server/dist/public/client-coagulacion/
```
5. Se entra en la carpeta client
```
[root@appmedicas ~]# cd coagulacion/client/
```
6. Se instalan las dependencias
```
[root@appmedicas client]# npm install
```
7. Se puede consultar en package.json del proyecto donde como se compila el proyecto.
```
[root@appmedicas client]# npm run build
```
8. Una vez compilado, se obtiene la carpeta dist/client-coagulacion/, entramos y copiamos todo lo que contiene a la carpeta que hemos creado en el punto 4.
```
[root@appmedicas client-coagulacion]# sudo cp -r . ~/coagulacion/server/dist/public/client-coagulacion/
```

9. Se debe ir a la carpeta

```
[root@appmedicas ~]# cd coagulacion/server/dist/
```

Donde se puede editar el archivo config.json (datos proporcionados al tutor)

```
[root@appmedicas dist]# sudo nano config.json
```

10. Se entra en la carpeta src

```
[root@appmedicas dist]# cd src/
```

11. Para poner el archivo principal en el administrador de procesos PM2

```
[root@appmedicas src]# pm2 start index.js --name api-coagulacion:3008
```

12. Se observa que ya está siendo escuchado

id	name	namespace	version	mode	pid	uptime	↺	status	cpu	mem	user
0	api-coagulacion:3008	default	1.0.0	fork	61052	110m	188	online	0%	76.2mb	root

13. Guardamos

```
[root@appmedicas src]# pm2 save
```

REFERENCIAS

[1] https://es.wikipedia.org/wiki/JSON_Web_Token

[2] https://es.wikipedia.org/wiki/Angular_(framework)

[3] https://angular.io/

[4] https://nodejs.org/es/

[5] https://es.wikipedia.org/wiki/Node.js

[6] https://www.youtube.com/watch?v=xJzzu7MVZXw

[7] https://developer.mozilla.org/es/docs/Learn/Server-side/Express_Nodejs/Introduction

[8] https://es.wikipedia.org/wiki/International_normalized_ratio

[9] https://es.wikipedia.org/wiki/Tiempo_de_protrombina

[10] https://jwt.io/

[11] https://programacionymas.com/blog/jwt-vs-cookies-y-sesiones

[12] https://angular.io/

[13] https://es.wikipedia.org/wiki/Angular_(framework)

[14] https://es.wikipedia.org/wiki/Secure_Shell

[15] https://www.hostinger.es/tutoriales/instalar-mysql-centos-7

[16] https://dev.mysql.com/downloads/repo/yum/

[17] https://matthiashoys.wordpress.com/2020/01/15/how-to-upgrade-node-js-from-v6-to-v12-on-centos-linux-7/

[18] https://github.com/nodesource/distributions

[19] https://computingforgeeks.com/how-to-install-bash-5-on-centos-linux/

[20] https://www.npmjs.com/package/pm2

[21] https://www.tecmint.com/install-pm2-to-run-nodejs-apps-on-linux-server/

[22] https://gitlab.com/

[23] https://firebase.google.com/

[24] https://es.wikipedia.org/wiki/Firebase

GLOSARIO

API: Application Programming Interface

REST: Representational State Transfer

NPM: Node Package Manager

XML: eXtensible Markup Language

JSON: JavaScript Object Notation

HTTP: Hypertext Transfer Protocol

HTML: HyperText Markup Language

PHP: Hypertext Pre-Processor

JWT: JSON Web Token

JSON: JavaScript Object Notation

IETF: Internet Engineering Task Force

RFC: Request for Comments

SPA: Single Page Application

PWA: Progressive Web Application

URI: Uniform resource identifier

INR: International normalized ratio

RAM: Reacciones adversas al medicamento

Nid: número de identificador

HMACSHA256 Código de autenticación de mensajes cifrados con el algoritmo hash 256 bits

Bash: Bourne-again ***shell***

PM2: Process Manager 2

FCM: Firebase Cloud Messaging

ANEXOS

ANEXO A: DISEÑO Y DISTRIBUCIÓN DE LAS PANTALLAS DE LA APLICACIÓN WEB

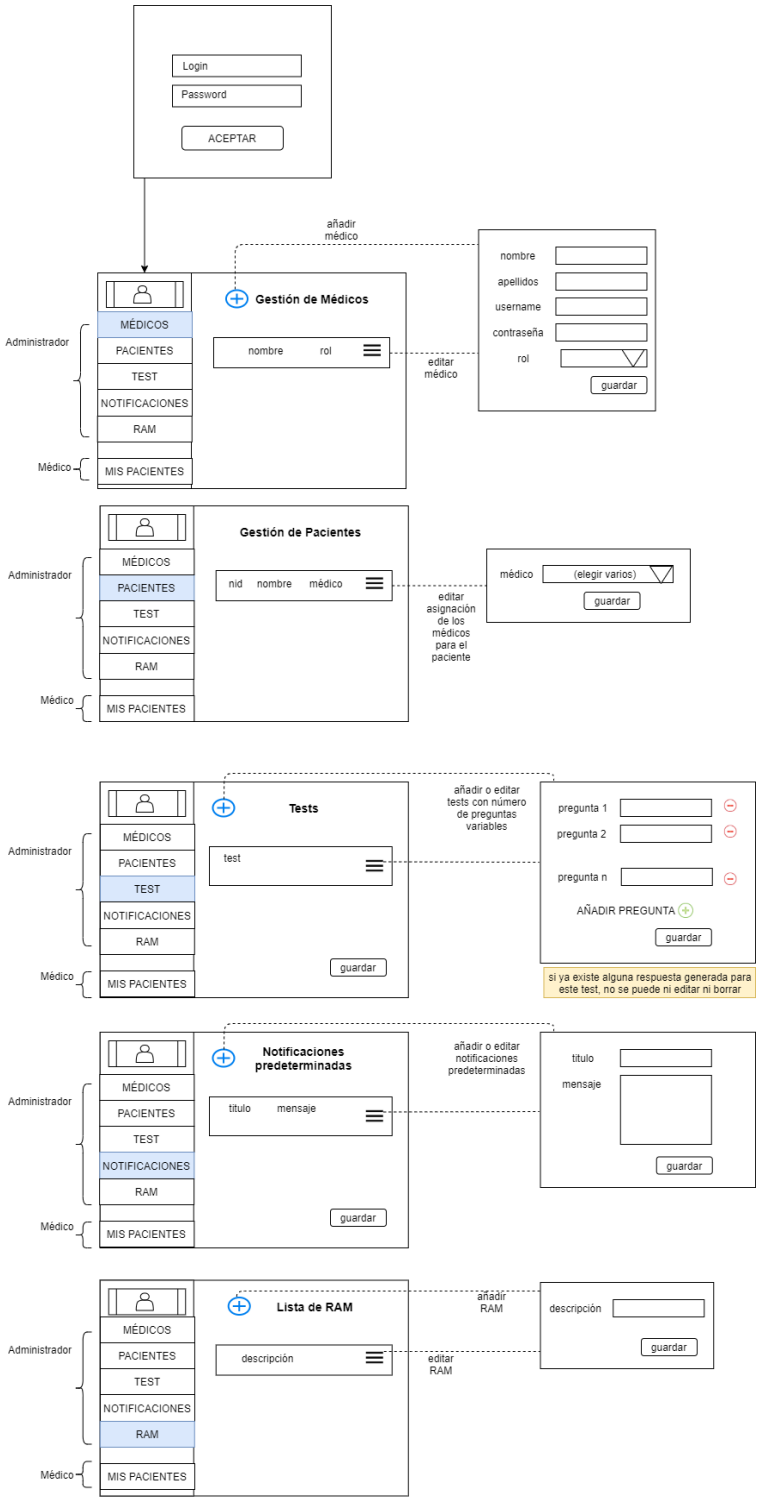


Figura Anexo-1. StoryBoard del módulo de Administrador de la aplicación

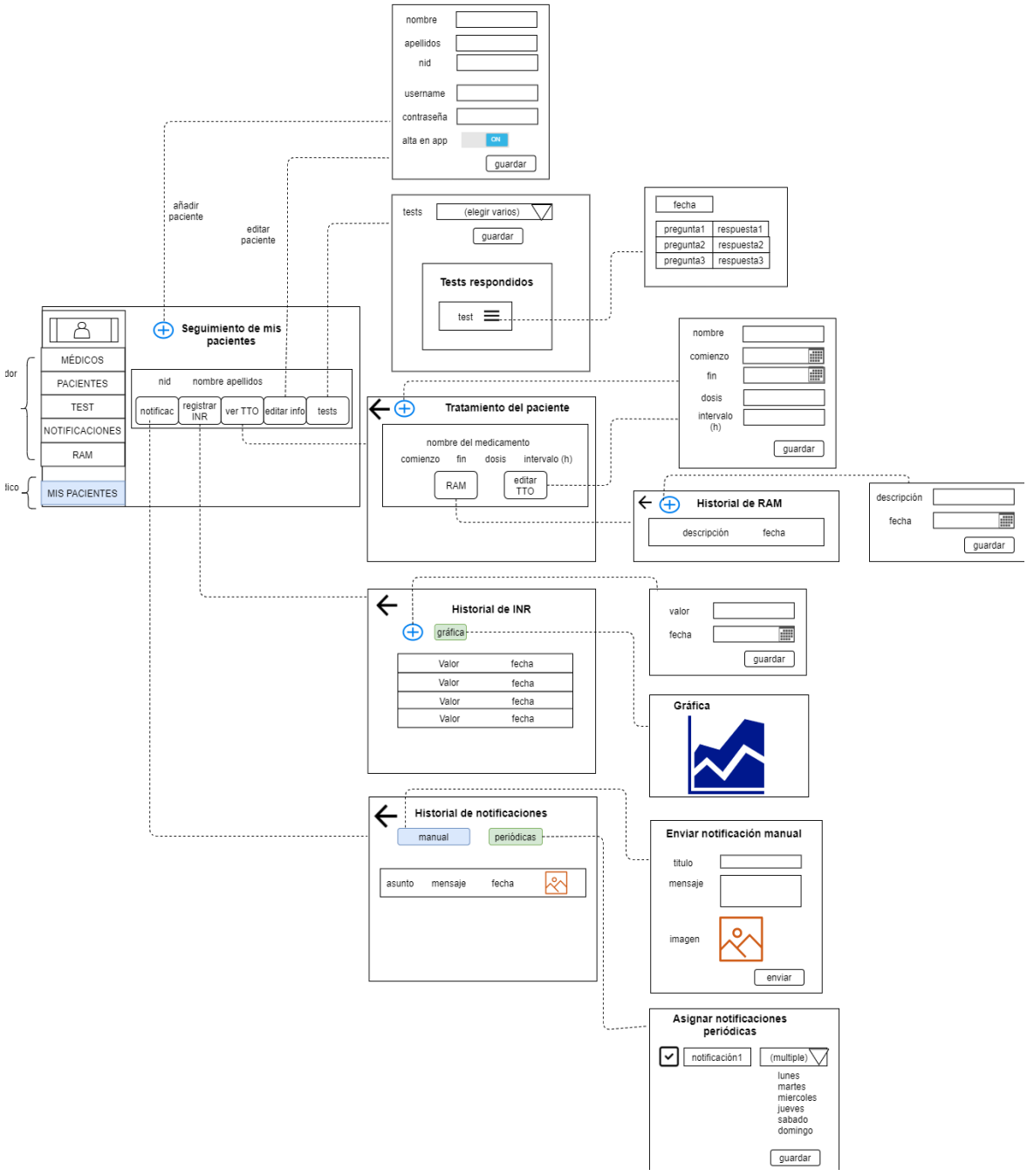


Figura Anexo-2. StoryBoard del módulo de Usuario de la aplicación

ANEXO B: MODELO DE DATOS

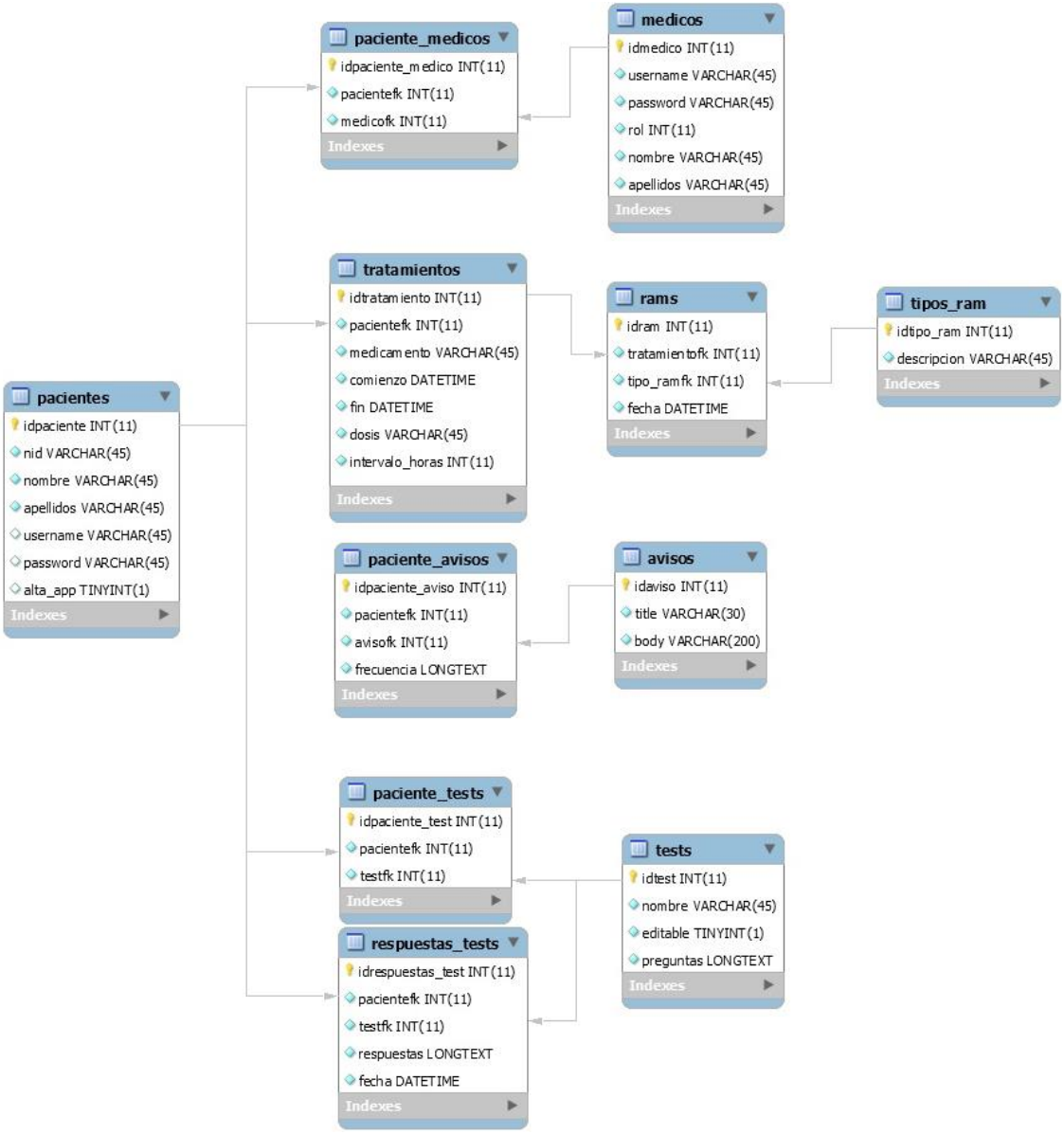


Figura Anexo-3. Modelo de datos I

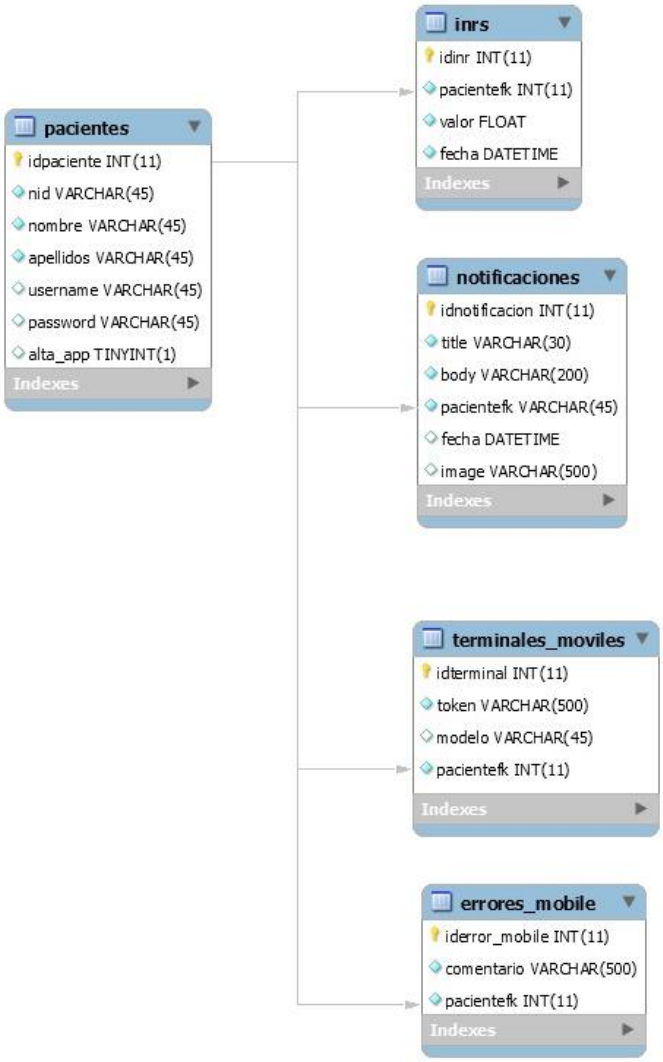


Figura Anexo-4. Modelo de datos II

Nota: Estas dos tablas registran cada acción que se realiza tanto en la aplicación web como en la aplicación móvil. Solo sirven para fines estadísticos.

Figura Anexo-5. Modelo de datos III